# A interligação do Visual C++ com as FPGAs da XILINX®

Valter Silva, Frederico Santos, Valery Sklyarov

***Resumo* – Este artigo apresenta uma aproximação possível para o desenho e simulação de sistemas usando FPGAs baseados em componentes que comunicam entre si e são implementados em software ou hardware. Esta técnica é muito útil para muitas aplicações práticas e em particular para o desenvolvimento de ferramentas do tipo Hardware/Software Codesign. É aqui apresentada a primeira experiência deste tipo. A ideia principal é apresentar uma abordagem efeciente e simples, e suas vantagens para uso didáctico e em sistemas reais. O desenvolvimento foi feito com a ajuda do Xilinx Foundation Software e o VHDL.**

***Abstract* – The paper presents an approach to the design and simulation of FPGA-targeted systems based on communicating components that are implemented both in software and in hardware. Such a technique is very useful for many practical applications and makes possible in particular to develop efficient tools for hardware/software codesign. We presents here just the first experience in this area. The primary idea was to demonstrate an effectiveness and relative simplicity of such an approach and its advantages for both educational process and real design problems. They have been developed with the aid of Xilinx Foundation Software and VHDL.**

## I. INTRODUÇÃO

O trabalho aqui descrito tem como base as disciplinas de opção «Engenharia de Software» e «Sistemas Digitais Avançados». O leccionado na primeira disciplina providencia o suporte para a implementação da interface gráfica e a modelação em *software* dos componentes de *hardware*. Isto foi desenvolvido recorrendo ao Visual C++. É importante que esta interface permita analisar diversos modos de funcionamento de uma maneira amigável. O leccionado na segunda disciplina forneceu bases para a implementação dos componentes de *hardware* baseados numa FPGA e as ferramentas para a comunicação entre o *software* e o *hardware*.

Os circuitos reconfiguráveis, tal como as FPGAs (*Field Programmable Gate Array*), permitem explorar novas técnicas de desenho de circuitos digitais. Normalmente estes circuitos reconfiguráveis permitem que se organize o processo de desenho nos seguintes passos:

1. Construir o circuito.
2. Simular e analisar o circuito por forma a verificar se satisfaz os requisitos.
3. Repetir os passos 1 e 2 até que todos os requisitos se verifiquem.

Existem dispositivos que suportam este tipo de desenho, como por exemplo *Development Boards* (placas de desenvolvimento) com carácter reconfigurável, tal como a XS40, Xstend e FireFly®. O desenho do circuito é feito usando o CAD especifico para cada FPGA, tal como o Xilinx Foundation Software, Synopsys e ViewLogic.

Para alguns sistemas pode-se realizar uma parte em FPGA que comunica com outros componentes do sistema, que podem ser digitais, mecânicos, ópticos, etc. Nalguns casos é importante testar o sistema completo e especificar fontreiras entre os componentes. Uma solução possivel para este problema é a simulação conjunta do *hardware* implementado na FPGA com o restante *hardware/software* pertencentes ao mesmo sistema. Este artigo apresenta os primeiros testes e resultados práticos nesta área.

Hoje em dia os circuitos digitais podem ser construídos por forma a lidarem com algum acontecimento físico, podem ser usados para controlarem alguma coisa como robots ou processos industriais. Pode-se visualizar a forma como os acontecimentos são tratados por exemplo com diagramas mecânicos, ou diagramas que ilustrem a cinemática de um sistema. Estes diagramas podem ser incorporados numa interface gráfica que permita de uma forma simples a verificação do funcionamento do sistema.

Este artigo apresenta um exemplo de uma simulação conjunta, para modelar e controlar os movimentos de uma plotter. Alguns algoritmos triviais de controlo da plotter são implementados numa FPGA XC4010XL da Xilinx®. Todos os movimentos da plotter são apresentados num PC, permitindo assim verificar visualmente o resultado do controlo efectuado pelo algoritmo implementado em *hardware* no FPGA.

Como este artigo é basicamente orientado para os estudantes, vamos explicar algumas funcionalidades básicas das FPGAs.

Uma FPGA (*Field Programmable Gate Array*), é um dispositivo constituido por células de lógica programável, chamadas CLBs (*Configurable Logic Blocks*). Todas as CLBs estão interligadas entre si por uma complexa teia de ligação também configurável. Tanto as CLBs como as ligações entre elas são programadas com recurso a RAM (*Random Acess Memory*) e logo pode-se reconfigurar o circuito um número ilimitado de vezes[1]. Desta característica deriva o facto de ser muito útil em projectos onde o *hardware* é alterado com muita frequência.

As FPGAs são uma tecnologia em grande expansão, cada vez mais usada e muito versátil. O seu uso é simples permitindo realizar sistemas digitais com elevado número de *gates* num único *chip* de pequenas dimensões. Beneficiam ainda da utilização de tecnologia CMOS VLSI, podendo atingir actualmente frequências de relógio de 80 Mhz.

A FPGA usada, XC4010XL da Xilinx, tem uma matriz de 20x20 CLBs, o que prefaz 400 CLBs[1].

A programação destas FPGAs, pode ser feita usando um *software* específico da marca, que permite utilizar esquemáticos, diagramas de estados ou mesmo linguagem VHDL.

O VHDL (*VHSIC Hardware Description Language*) é uma linguagem para descrever circuitos digitais[6], e nasce devido à necessidade de uma linguagem *standard* de descrição de circuitos digitais, no âmbito de um ambicioso programa de desenvolvimento de circuitos digitais de alta velocodade e respectivas tecnologias de projecto (*VHSIC – Very Higth Speed Integrated Circuit*).

Esta linguagem serve para descrever a estrutura e funcionalidade de um circuito digital. Fazendo-nos pensar no circuito de uma forma abstracta, não havendo necessidade de preocupação com a tecnologia específica de implementação do circuito lógico em *hardware*.

A FPGA utilizada está integrada numa *Development Board* XS40, da mesma marca, que possui, alguns mecanismos de interacção com o utilizador, como displays de 7 segmentos ou botões de pressão. Está ainda dotada de um micro-processador 8051 da Siemens®, e 32Kb de SRAM (*Static Ramdom Access Memory*). Esta placa pode ainda estar integrada noutra *board XStend* para se obter mais funcionalidades, como por exemplo 64Kb de SRAM adicionais e área disponível para ligação de componentes lógicos adicionais.

Estes *Development Boards*, servem essencialmente para, construir pequenos exemplos de demonstração devido à sua versatilidade e disponibilidade de dispositivos de interacção com o utilizador.

A figura 1 mostra a estrutura básica e a comunicação entre o *software* e o *hardware*.

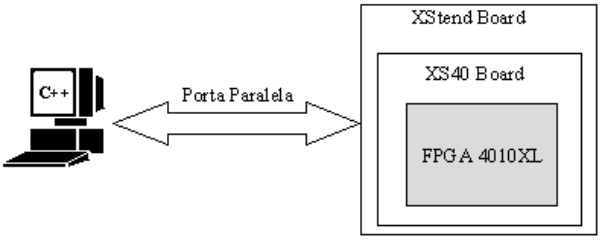


Fig. 1 – O esquema geral

O sistema é constituído por duas partes: uma parte é *software* desenvolvido num PC, usando a linguagem C++; e outra complementa esta, ou seja efectua o controlo em *hardware* na FPGA.

## II. DETALHES DE IMPLEMENTAÇÃO

O objectivo do trabalho foi simular o funcionamento de um traçador de gráficos (Plotter) em que os movimentos e a caneta a utilizar são controlados pelo *hardware* ou caso o utilizador pretenda, manualmente.

A velocidade de execução é configurável, podendo variar de 1 a 10 movimentos por segundo (modo automático), ou em alternativa passo a passo (modo manual).

Tal como se pode ver na figura 2, a Plotter é constituida por um braço e uma cabeça que contém as canetas de escrita. O braço movimenta-se verticalmente, enquanto a cabeça tem movimentos horizontais.

Na figura seguinte, figura 2, pode-se ver a Plotter desenvolvida. Existem quatro sensores que indicam as posições extremas, isto é dois sensores para movimentos horizontais (esquerda e direita), e dois verticais (cima e baixo).

Mesmo este exemplo simples pode ser extendido por forma a analizar diferentes modos de funcionamento, tal como:

1. Testar diferentes estados iniciais.
2. Trabalhar em diferentes sistemas de coordenadas.
3. Testar diferentes algoritmos de desenho (para imagens 2D e 3D).

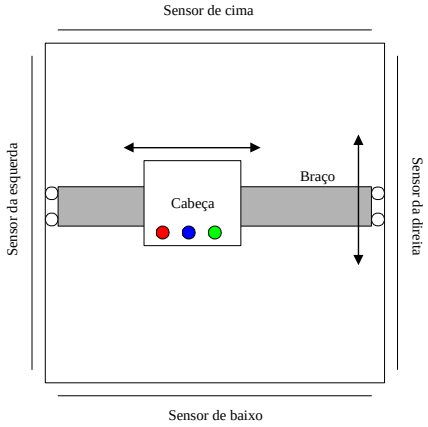


Fig. 2 – Plotter emulada

De uma forma geral pode-se dizer que o problema se divide em três partes:

1. Mecanismos de comunicação entre o *software* e o *hardware*.
2. Parte de *software* e interface com o utilizador.
3. Parte de *hardware*.

Todas estas parte serão descritas mais à frente em mais detalhe.

## III. A COMUNICAÇÃO ENTRE O PC E A FPGA

*A Interface de entrada e saída no PC*

A comunicação com a FPGA é feita através da porta paralela (LPT1), utilizando um cabo onde todos os pinos estão ligados ao pino com o mesmo número na outra ponta do cabo.

Para a comunicação com a FPGA, pode ser construída uma classe, mostrada na figura 3, com duas funções, uma para envio (função `send`) de valores para a FPGA e outra para a recepção (função `receive`) de informação proveniente desta.

```
class fpga
{
    int lpt1_address;
    public:
    fpga(){lpt1_address=*((int *) (0x408));};
    ~fpga(){};
    void send (unsigned char);
    int receive (void);
}
```

Fig. 3 – A classe FPGA

O endereço do registo *Data Address* da porta paralela, pode-se saber, indo ler o endereço 0x408 (este endereço é igual em todos os PCs), isto pode ser feito no constructor da classe FPGA[2][3].

Esta classe terá então uma variável que serve para armazenar o endereço do *Data Adress* da porta paralela e duas funções para se fazer o envio e a recepção de informação.

O constructor é definido *inline*[3][4], enquanto que as duas funções são definidas *outline* [3][4] e serão explicadas mais adiante.

- A função `void send ( unsigned char )`

Esta função serve para enviar para a FPGA um valor de 8 *bits.*

Como na *Development Board* onde a FPGA está inserida, os dois *bits* menos significativos são negados, é necessário negá-los nesta função para que na FPGA se tenha exactamente o mesmo valor enviado. A estrutura do *byte* é mostrada na figura 4.


Fig. 4 – O byte a enviar

O endereço, para onde é necessário fazer a escrita, é o *Data Address* do controlador da porta paralela.

Sendo assim o código utilizado para o envio de um *byte* é o mostrado na figura 5.

```
void fpga::send(unsigned int value)
{
  outportb(lpt1_address,
           (~value&0x03)|(value & 0xfc));
}
```

Fig. 5 – A função send

Este código, foi desenvolvido para o compilador Borland® C++, para se compilar com o Microsoft® Visual C++, deve-se substituir a função `outportb`, pela função `_outp`.

- A função `unsigned receive( void )`

Quando se pretende receber um *byte*, os únicos *bits* relevantes, são os bits 3,4,5,6, apresentados na figura 6, que são lidos do *status register* do controlador da porta paralela, situado no endereço *Data Address* + 1.

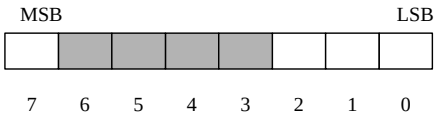


Fig. 6 – Os bits importantes

É necesário fazer um *and bit* a *bit* com o valor 0x78, e em seguida fazer um *shift* para a direita de 3 *bits*. O código desta função é mostrado em seguida.

```
unsigned fpga::receive(void)
{
  int temp;

  temp=inportb(lpt1_address+1);
  temp&=0x78;
  temp>>=3;
  return temp;
}
```

Fig. 7 – A função receive

Tal como para a função `send ( unsigned int )`, também nesta função é necessário alterar a função `inportb` por `_inp`, para se compilar com o Microsoft Visual C++.

Assim sendo, o número de bits disponíveis para envio de informação do PC para a FPGA é de 8, enquanto que no sentido inverso estão disponíveis apenas 4 *bits*, tal como a figura 8 apresenta.

Na FPGA, é necessário providenciar uma interface com o exterior, que é feita com recurso a componentes que as bibliotecas do *Software* de programação da FPGA disponiblizam. Esta interface será explicada no próximo ítem neste artigo.

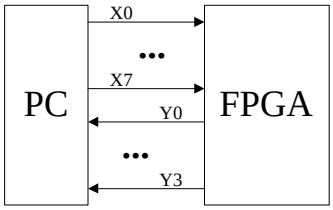


Fig. 8 – As comunicações

*A interface de entrada e saída na FPGA*

A programação da FPGA, tal com já foi dito é feita utilizando um programa específico da Xilinx.

O primeiro passo a executar é especificar o circuito, onde se pode recorrer a esquemáticos, à linguagem VHDL ou mesmo a diagramas de estados. Em seguida e nos dois primeiros casos é necessário compilar a descrição. Pode-se então executar uma simulação temporal. Depois de se implementar o trabalho é gerado um ficheiro, com a extensão *.bit*, que serve para se programar a FPGA. Em [8] encontram-se mais pormenores sobre a programação da FPGA e suas funcionalidades.

Esta programação é feita recorrendo ao programa *xsload*, que é executado em MS-DOS, que faz o *download*, através da porta paralela, do ficheiro *.bit* para a FPGA. A figura 9 representa estes passos.

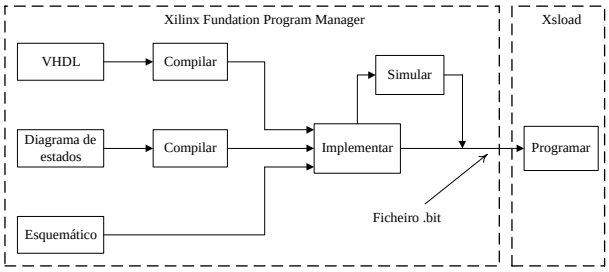


Fig. 9 – A programação da FPGA

Quando se usam diagramas de estados, é também necessário compilar, porque o programa cria código VHDL, a partir desse referido diagrama, processo este transparente para o utilizador.

No PC, foram desenvolvidas funções (mostradas em items anteriores) para providenciar a comunicação com a FPGA, havendo então necessidade de implementar um sistema que disponibliza a interface com a porta paralela na FPGA.

Os pinos de entrada aos quais a porta pararela está ligada, são do menos para o mais significatiovo respectivamente; 44, 45, 46, 47, 48, 49, 32, 24.

Os pinos de saída aos quais a porta série está ligada são do menos para o mais significativo; 70, 77, 66, 69.

Na figura 10, estão representados os terminais de entrada e de saída necessários para providenciar a comunicação com a porta paralela, podendo então ser criada uma macro, que torna estes pormenores transparentes para o utilizador. Por baixo de cada *Pad* está apresentado o respectivo pino da FPGA que lhe está associado. Para se ter acesso a esse pino é necessário, utilizar os *Pads* da figura 10, IPAD, ou OPAD, consoante se desejar entradas ou saídas, com LOC igual ao pino físico da FPGA.

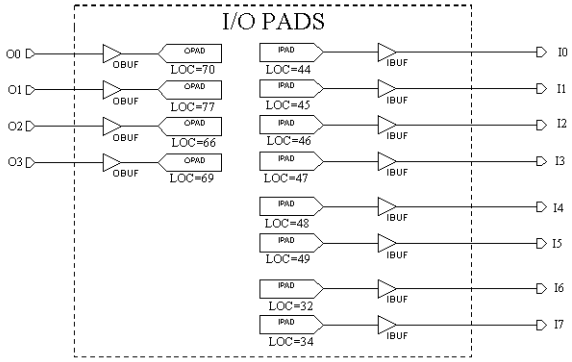


Fig. 10 – terminais de entrada e saída

Para cada *bit* utiliza-se um *buffer*, de entrada ou saída consoante o tipo deste.

Esta macro é realizada com recurso ao editor de esquemáticos do Xilinx Foundation Project Manager.

Com o explicado anteriormente, é possivel construir qualquer projecto deste tipo, sendo apenas necessário definir a forma como se envia e recebe informção, ou seja, apenas é necessário definir um protocolo de comunicação.

## IV. A PROGRAMAÇÃO DA FPGA

O programa desenvolvido para a FPGA recorre essencialmente a diagrama de estados, onde construimos uma máquina de estados finitos (FSM – *finite state machine*).

De modo a ser mais perceptivel, a explicação de como se executa a programação da FPGA, iremos explicar inicialmente o protocolo utilizado para a comunicação entre o PC e a FPGA.

*O protocolo de comunicação*

Como foi mencionado anteriormente a Plotter dispunha de 4 sensores, ou seja um para cada posição extrema. O estado destes sensores são enviados para a FPGA nos 4 *bits* menos significativos dos 8 *bits* disponíveis.

Do PC para a FPGA enviamos ainda um sinal de relógio, gerado no PC, para podermos controlar a velocidade de execução da Plotter. Este sinal de relógio serve também de *clock* para a nossa máquina de estados finitos.

Utiliza-se ainda outro *bit* para multiplexar as saídas da máquina de estados, já que estas saídas são 7 e apenas se dispõe de 4 *bits* de saída da FPGA.

Os bits de entrada na FPGA são:

- I0 – sensor de baixo.
- I1 – sensor da esquerda.
- I2 – sensor de cima.
- I3 – sensor da direita.
- I4 – *Clock*.
- I5 – *bit* de multiplexagem.

Os bits de saída da máquina de estados são:

- Y1 – liga motores verticais.
- Y2 – liga motores horizontais.
- Y3 – 1 – sentido para cima; 0 – sentido para baixo.
- Y4 – 1 – sentido para a direita; 0 – sentido para a esquerda.
- Y5 – escreve com a caneta vermelha.
- Y6 – escreve com caneta verde.
- Y7 – escreve com caneta azul.

O Reset à máquina de estados é efectuado aquando do envio para a FPGA dos quatro sensores simultaneamente activos, o que é fisicamente impossível.

O esquema geral do harware desenvolvido para programar a FPGA é mostrado na figura 11.

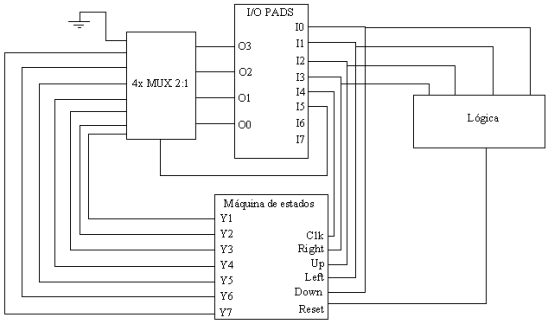


Fig. 11 – O esquema geral

Assim sendo quando o *bit* de controlo dos multiplexers tem o valor ‘0’ à saída deste (ou à entrada dos I/O PADS) obtém-se os sinais: Y1;Y2;Y3; Y4.

Se, por outro lado I5, tiver o valor ‘1’, à saída dos multiplexers obtem-se os sinais: Y5; Y6; Y7; Gnd.

Assim, no *software* desenvolvido no PC, em C++, é possível controlar qual destes dois últimos conjuntos de sinais se pretende.

Para o *clock* é necessário implementar um atraso, que se torna imprescíndivel, porque as entradas da máquina de estados têm que estar estáveis antes do ocorrer o evento de *clock* para uma possível transição de estado. Este atraso não está representado na figura 9, e será discutido mais adiante.

*O circuito de reset*

Tal como já foi referido, é necessário fazer *reset* à máquina de estados.

Para se poder ter controlo sobre a máquina de estados, implementou-se o circuito de forma a no programa que corre no PC (desenvolvido em Visual C++) fazer este *reset*[5].

Isto é conseguido enviando para a FPGA a situação em que todos os sensores estão activos. Optou-se por esta situação por ser fisicamente impossível.

Quando todos os sinais (*left*, *right*, *up*, *down*), estiverem a ‘1’ (activos) é necessário fazer *reset* à máquina de estados, em qualquer outra situação, não se pode fazer *reset*.

Então a lógica necessária é um *and* dos quatro sinais, isto para quando, todos tomarem o valor ‘1’ ser feito o *Reset* à máquina de estados.

Isto é mostrado na figura 12.

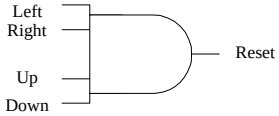


Fig. 12 – Circuito de *reset*

*O atraso de relógio*

Tal como já foi explicado é necessário implementar um atraso de relógio para que as entradas da máquina de estados estejam estáveis antes da transição do relógio, isto porque o sinal de relógio é enviado para a FPGA ao mesmo tempo que os restantes sinais.

Como uma máquina de estados é implementada utilizando *Flip-Flops*, os sinais têm de obedecer a uma determinada temporização[1].

O atraso implementado pode ser um atraso pequeno já que se está a lidar com tecnologia rápida.

Este atraso de 250 nS, é construído com recurso a 2 *Flip-Flops*[5], ligados em cascata, tal como é mostrado na figura 13.

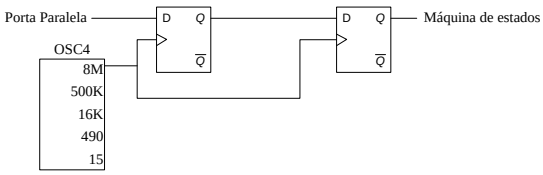


Fig. 13 – O atraso de relógio

*A máquina de estados finitos*

A máquina de estados finitos é a base de todo o trabalho desenvolvido para a FPGA no nosso projecto.

Foram desenvolvidas duas máquinas de estados, uma para fazer regressar a cabeça da Plotter ao canto inferior esquerdo, ponto (0,0), e outra que além de fazer regressar a Plotter ao ponto (0,0), desenha três figuras geométricas, em diferentes cores, e regressa ao canto inferior esquerdo, mas neste caso é efectuada a contagem do espaço percorrido e não através dos seus sensores.

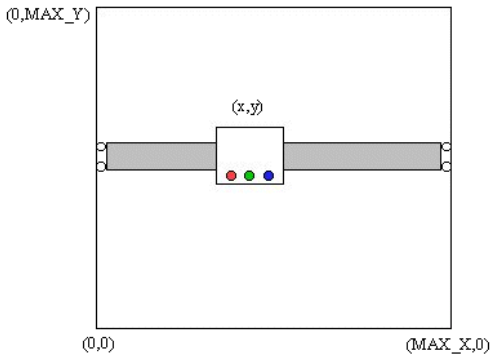


Fig. 14 – A escala da Plotter

Para a máquina de estados que apenas retorna a Plotter ao canto inferior esquerdo podemos definir os seguintes requisitos:

- No caso da cabeça da Plotter estar numa posição (x,y), com x≠0 e y≠0 tem de se deslocar na diagonal para baixo e para a esquerda.
- Se estiver numa posição (0,y), tem de se movimentar no sentido de y decrescente.
- Se estiver numa posição (x,0), tem que se movimentar no sentido de x decrescente.
- Se estiver na posição (0,0), deve mantê-la.

Naturalmente, temos então que definir 4 estados:

- Estado que permite à Plotter movimentar-se em x e y descendente.
- Estado que faz que a Plotter se movimente em y descendente, mantendo x constante.
- Estado que faz que a Plotter se movimente em x descendente, mantendo y constante.
- Um estado que pare a Plotter.

Em todos os estados anteriores todas as canetas de escrita devem ser desactivadas.

É necessário definir outro estado; estado esse para o qual a máquina vai quando é feito o *reset*, e é a partir da posição em que se encontra nesse momento que inicia a sua marcha.

A máquina de estados mostrada na figura 15 implementa o regresso ao ponto (0,0) da Plotter.

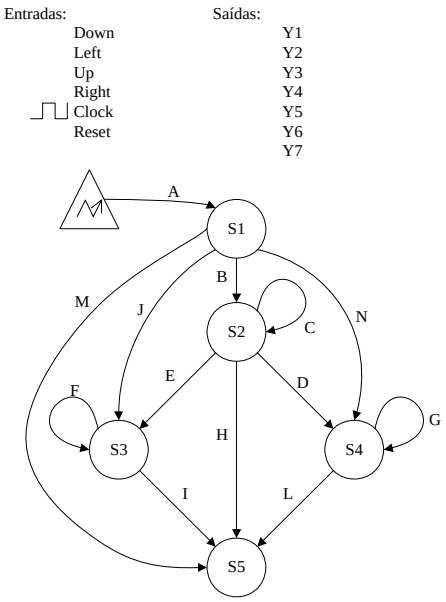


Fig. 15 – O diagrama de estados

Cada transição representada na figura, tem uma condição associada, que é a condição para a qual, ao evento do relógio, se dá a mudança de estados.

Esta máquina de estados, está definida, como sendo sensível à transição ascendente do sinal de relógio (*Clock*). O editor de diagramas de estados, permite que se defina outro tipo sensibilidade para o relógio, como por exemplo sensível à transição descendente.

O estados representados na figura, têm o seguinte significado:

- S1 - «Desliga» todas as canetas de escrita, pára todos os motores, o que corresponde aos sinais de saída de Y1 a Y7 todos a '0'.
- S2 - «Desliga» todas as canetas de escrita, anda no sentido de 'y' decrescente e 'x' decrescente, o que corresponde aos sinais Y1 e Y2 a '1' e os restantes a '0'.
- S3 - «Desliga» todas as canetas de escrita, anda no sentido 'y' decrescente, com 'x' constante, o que corresponde ao sinal Y1='1' e os restantes a '0'.
- S4 - «Desliga» todas as canetas de escrita, anda no sentido 'x' decrescente, com 'y' constante, o que significa Y2='1' e os restantes a '0'.
- S5 - «Desliga» todas as canetas de escrita, pára ambos os motores, é atingido o ponto (0,0), o que quer dizer todos os sinais de saída a '0'.

O estado S1 é sempre o primeiro estado atingido, pois é o estado para o qual a máquina de estados vai ao fazer-se *reset*. O *reset* é feito sempre que se pretende atingir o ponto (0,0), sendo a FPGA a controlar.

O *reset* é assíncrono, sensível às transições, assim, e como a transição «A» corresponde a uma condição de

Reset='1', a máquina de estados faz o reset quando existe uma transição positiva do sinal de entrada *Reset*.

Todas as acções são executadas logo que o respectivo estado é atingido (*entry action*). O editor permite ainda, que se definam acções para as transições (*transition action*) de estado, assim como para a saída (*exit action*) de estado e para a permanência nesse estado (*state action*).

Para que a Plotter, atinga o ponto (0,0), os estados de S1 até S5 são percorridos, consoante a situação em que a cabeça da Plotter se encontra. Assim e para os diferentes casos:

- Para x=0 e y=0 : S1→S5
- Para x=y : S1→S2→S5
- Para x>y com x≠0 e y≠0 : S1→S2→S4→S5
- Para y>x com x≠0 e y≠0 : S1→S2→S3→S5
- Para x=0 e y≠0 : S1→S3→S5
- Para y=0 e x≠0 : S1→S4→S5

Assim as condições de passagem de estado, representadas na figura 15 são:

- A ➔ Reset='1'
- B ➔ Down='0' and Left='0'
- C ➔ Down='0' and Left='0'
- D ➔ Down='1' and Left='0'
- E ➔ Down='0' and Left='1'
- F ➔ Down='0'
- G ➔ Left='0'
- H ➔ Down='1' and Left ='1'
- I ➔ Down='1'
- J ➔ Down='0' and Left='1'
- L ➔ Left='1'
- M ➔ Down='1' and Left='1'
- N ➔ Left='1' and Down='0'

*Um exemplo*

Para o caso da cabeça da Plotter estar numa posição x=y com x≠0 e y≠0 os estados percorridos são os apresentados a seguir correspondentes à transição ascendente do *Clock*.

- É feito o *reset* à máquina de estados, o estado atingido é o S1.
- Como x≠0, o sensor da esquerda não detecta a cabeça da Plotter e logo Left='0'; y≠0 e logo também Down='0'.
- Das três possíveis transições, a executada é a «B», porque Down='0' e Left='0'.
- A máquina de estados mantem-se neste estado, efectuando a transição «C» até que x=0 ou y=0, que acontece no mesmo instante já que x=y. Quando x=0 e y=0, Left='1' e Down='1'.
- Como Left='1' e Down='1', dá-se a transição «H» e o estado final S5 é atingido, o que quer dizer que a Plotter chegou à posição (0,0).

*Desenhar 3 figuras geométricas*

Para se desenhar com a Plotter, através da máquina de estados, 3 figuras geométricas é necessário definir um estado por cada movimento que a Plotter irá efectuar[5].

Para tal defeniram-se duas variáveis contadoras para controlar os movimentos da Plotter, por forma a poder desenhar as figuras geométricas. Estas duas variáveis também foram utilizadas para que depois de se desenhar as 3 figuras, se possa fazer o regressar à posição (0,0) da cabeça da Plotter.

Uma outra variável, «Stop» é introduzida para que quando se acabe de desenhar o pretendido, se regresse ao estado S5, sem que se recomeçe a desenhar outra vez o já desenhado.

As 3 figuras geométricas são desenhadas a diferentes cores, é então necessário activar a caneta correcta. Por exemplo, para activar a caneta vermelha é necessário colocar o sinal Y5 a '1'.

Logo para se desenhar uma linha do ponto (0,0) ao ponto (10,0), usa-se o diagrama de estados da figura 17.

Assim, é necessário acrescentar a atribuição do valor '0' ao sinal «Stop», no estado S1.

No estado S5, é também necessário inicializar as variáveis contadoras, «Horizontal» e «Vertical».

As condições de transição de estado são:

- O ➔ Stop='0'
- P ➔ Horizontal<=10
- Q ➔ Horizontal>10

No estado S6, é necessário ligar o motor horizontal no sentido de x crescente, activar a caneta azul, incrementar a variável «Horizontal» e atribuir a «Stop» o valor '1'. Estas acções correspondem a colocar Y2,Y4 e Y7 a '1' além de `Stop:=1` e `Horizontal:=Horizontal+1`.

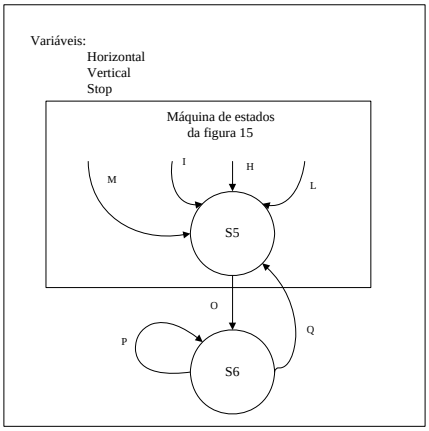


Fig. 17 – A máquina de estados para desenhar uma linha

*O uso de VHDL*

Poderia-se-ia ter desenvolvido a máquina de estados em VHDL, isto não foi feito devido à melhor adequação dos diagramas de estados para este caso.

Por este exemplo, ir-se-á demostrar que em VHDL se descreve não só o nível funcional, mas também o nível estrutural do nosso dispositivo.

O nível estrutural fica descrito, com a declaração da entidade (*entity*) [6]. Ou seja descreve a visão externa do dispositivo, com esta declaração da entidade, verifica-se que existem três entradas e uma saída, todas do tipo *bit*. Isto pode ser visto na figura 17.

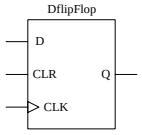


Fig. 17 - A entidade DfliFlop

O código VHDL para o *Flip-Flop* da figura 17, tipo D, com *Clock* e *Clear*, é o mostrado na figura 18.

```
entity DflipFlop is        -- Entidade
    Port (D,CLK,CLR: in bit;
          Q: out bit );

architecture ArcDflipFlop of DflipFlop is
    begin
    process (CLK,CLR)      -- Linha 6
    begin
        if (CLR='1') then
            Q<='0';        -- Linha 9
        elsif (CLK'event and CLK='1') then
            Q<=D;
        end if;
    end process;           -- Linha 13
end ArcDflipFlop;
```

Fig. 18 – Um Flip-Flop em VHDL

Por outro lado, a arquitectura (*architecture*), descreve a parte funcional do dispositivo, ou por outras palavras é a visão interna do mesmo. Cada entidade pode conter várias arquitecturas, correspondentes a diversas implementações da mesma função.

Deste exemplo pode-se reter uma diferença importante para as linguagens de programação convencionais, presente apenas nas linguagens de descrição de *hardware*, que é a atribuição de valores a sinais, como é o caso da linha 9, deste exemplo.

Como as máquinas de estados são traduzidas para VHDL, é necessário usar a sintaxe do VHDL, ou seja, neste caso, para atribuição de um valor a um sinal ( caso de Y1..Y7 ), usa-se o operador '<=', no caso de variáveis usa-se ':='.

Por exemplo, para a atribuição de um valor a um sinal usa-se:

```
Y2<='1';
```

Para uma variável usa-se:

```
Stop:=1;
```

A linha 6, indica que se terá de executar umas determinadas instruções, quando ocorrer uma mudança nos sinais CLK, CLR. Em seguida e através das instruções de `if`, consegue-se distinguir se foi uma transição descendente ou ascendente. Neste caso pretende-se as transições ascendentes.

Na linha 13, depois de todas as instruções serem executadas o processo suspende à espera que ocorra uma mudança nos sinais, ao qual é sensivel, neste caso CLR e CLK.

*Descrição duma máquina de estados simples usando VHDL*

Vamos dar um pequeno exemplo, de como se pode executar uma máquina de estados usando VHDL.

Os requisitos do nosso dispositivo, a que chamaremos XPTO são:

- A saída y, irá a '1', no próximo evento do relógio após a entrada x estar a '1'.
- Passados 10 eventos de relógio, a saída irá a '1', independentemente da entrada x, regressará a '0' no próximo evento de relógio.
- Quando for necessário, pode-se usar uma entrada de *Reset*, para colocar a saída a'0', independentemente do sinal de relógio.
- A saída regressa a '0', logo que a entrada regresse a '0'.

Com estes requisitos, já se pode definir o exterior do dispositivo XPTO mostrado na figura 19, e também declarar a entidade correspondente.

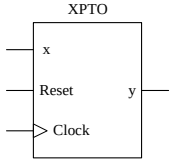


Fig. 19 - O dispositivo XPTO

Pode-se ainda dizer, que o sinal de *Reset*, será um sinal assíncrono, já que se deseja fazer *Reset* à máquina de estados independentemente do sinal de relógio.

A nossa máquina de estados terá, 2 estados, o primeiro a que corresponde a saída a '0', e o segundo a que corresponde a saída a '1'.

A nossa máquina de estados é mostrada na figura 21.

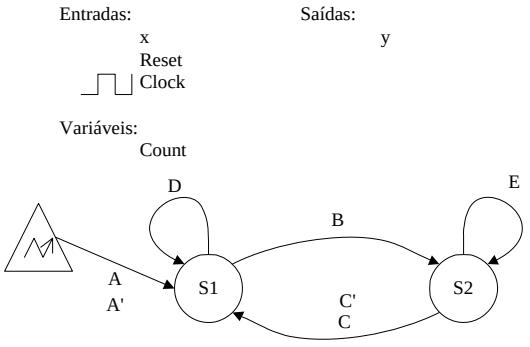


Fig. 20 – A máquina de estados XPTO

Na figura 20, «A'» e «C'», representam acções executadas, nas respectivas transicções.

As acções que se pretendem executar no estados S1 e S2, são acções que se executam assim que esse estado é atingido (*entry action*).

As acções são:

- Estado S1 ➔ colocar o y a '0' e incrementar o valor de «Count».
- Estado S2 ➔ colocar y a '1'.

As condições de transição de estado, são:

- A ➔ Reset='1'
- B ➔ x='1' ou Count>=10
- C ➔ x='0'
- D ➔ x='0' e Count<10
- E ➔ x='1'

As acções «A'» e «C'», são as duas iguais e correspondem a colocar a variável «Count» a zero, para se iniciar a nova contagem.

O código VHDL, para a referida máquina de estados, é o apresentado no anexo 1.

Tal como numa linguagem de programação normal, é possível definir tipos de dados, como é o caso do apresentado na linha 11.

A variável «Count» está definida na linha 16.

O processo definido na linha 15, é sensível à transição do sinal de relógio e do sinal de *Reset*.

Por exemplo, no caso de termos um *Reset* síncrono, com o sinal de relógio, o nosso processo seria apenas sensível ao sinal de relógio e teríamos também de alterar o programa por forma a que só fosse feito *Reset* quando ocorresse a transição positiva do relógio.

## V. O PROGRAMA PARA O PC

A programação do PC, foi feita com recurso ao Microsoft® Visual C++ 6.0.

Desenvolveu-se uma aplicação do tipo SDI (*Single Device Interface*)[7], onde é apresentada a Plotter, com os respectivos quatro sensores , apresentada na figura 21.

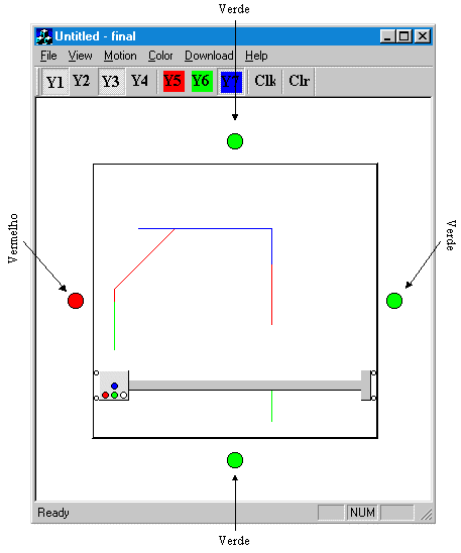


Fig. 21 – A aplicação desenvolvida

Foi construída uma barra de ferramentas (*toolbar*) de modo a ser mais fácil a interacção com o utilizador[5]. Nessa *toolbar* incluem-se os botões para controlo dos motores (Y1 a Y4), para controlo das canetas de escrita (Y5 a Y7), e ainda um botão de *Clock* e outro de *Clear*.

Os botões de Y1 a Y7, têm exactamente o mesmo significado dos sinais apresentados, no sub-título «O protocolo de comunicação» do capítulo «IV. A programação da FPGA» deste artigo.

Todos os movimentos da Plotter também podem ser alterados através do item *Motion* do menu.

O botão *clear* serve para limpar a zona de escrita da Plotter.

Os movimentos da Plotter, podem ser controlados de duas formas, ou pela FPGA, ou pelo utilizador.

Quando a Plotter está a ser controlada pela FPGA, os botões de Y1 a Y7, ficam desactivados, já que quem envia esses sinais é a FPGA, tal como já foi explicado.

Para a FPGA poder controlar a Plotter é necessário fazer o *Download* do ficheiro *.bit*, usando para tal a opção *Download* do menu. É então aberta uma janela de diálogo onde se digita o nome do ficheiro. Esta janela de diálogo por sua vez executa o programa *xsload.exe*, já referido.

A velocidade de execução é definida pelo utilizador, tanto no caso de ser ele a controlar os movimentos, como no caso de ser a FPGA.

Pode-se optar por uma execução passo a passo, ou por uma execução com velocidade automática, esta pode ir desde 1 até 10 passos por segundo.

Com uma execução passo a passo o botão *clock* é utilizado para executar cada passo, quando se usa uma velocidade automática esse botão fica inacessível.

O relógio necessário para a máquina de estados que funciona na FPGA é, no caso de execução passo a passo, feito pelo accionamento do botão Clk. No caso de uma velocidade automática gerado pelo PC, é feito consoante a velocidade escolhida.

Para se configurar a velocidade de execução escolhe-se *Motion→Speed* no menú. Aparece então uma janela de diálogo, apresentada na figura 22, onde, através de um *slider* se escolha a velocidade.

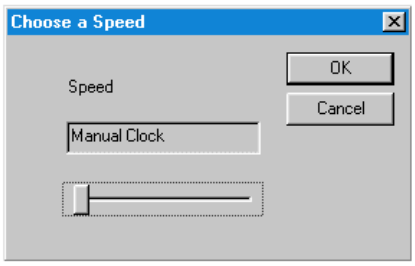


Fig. 22 – A escolha da velocidade

Como se pode ver pela figura 21, a aplicação apresenta os sensores, e o seu estado, através de um círculo. Quando vermelho o sensor está activo, a verde o sensor está inactivo. No caso da figura 21, a cabeça da Plotter está na posição mais à esquerda possível e logo esse sensor está activo.

O estado destes sensores é enviado para a FPGA, quando a Plotter é controlada por esta, tal como foi explicado no sub-titúlo «O protocolo de comunicação» do capítulo «IV. A programação da FPGA» deste artigo.

## VI. CONCLUSÕES

Deste artigo deve-se reter, essencialmente, como de uma forma simples se constrói algum *hardware* numa FPGA, recorrendo a diversos métodos de desenho e se desenvolve *software* que controla e comunica com esse *hardware*. Todas as formas de construção aqui discutidas, não são de forma alguma uma regra, já que as FPGAs, permitem muitos graus de liberdade, tal como foi exemplificado para o desenho de uma máquina de estados.

A interligação com o C++, ou em particular com uma aplicação gráfica desenvolvida em Visual C++, é simples, sendo apenas essencial, as interfaces de entrada e saída apresentadas para a FPGA e para o PC.

## AGRADECIMENTOS

Gostariamos de agradecer muito ao Professor António Ferrari e à Lurdes Sousa pelas suas colaborações neste artigo.

## REFERÊNCIAS

[1] "The Programmable Logic Data Book", Xilinx Inc , 1998

[2] Valter Silva e Frederico Santos, "Final Task of Object Oriented Programming and Advanced Digital Systems – Structure", Fevereiro 1999.

[3] Bjarne Straustroup, "The C++ programming Language", Third Edition, Addison Wesley, 1998.

[4] Valery Sklyarov, "Understanding and Low Level Implementation Basic OOP Construction", Electrónica e Telecomunicações, Vol. 1, Nº 7, Janeiro de 1997, 729-738

[5] Valter Silva e Frederico Santos, "Trabalho Final de Engenharia de Software - Relatório", Julho 1999.

[6] Peter J. Ashenden, "The Designer's Guide to VHDL", Morgan Kaufmann, 1996.

[7] Steven Holzner, "Visual C++ 6 in Record Time", Sybex, 1999.

[8] Iouliia Skliarova, António B. Ferrari, "Projecto e Implementação de um subconjunto da arquitectura MIPS16 com base em FPGA XC4010XL", Electrónica e Telecomunicações, Vol. 2, Nº 6, Setembro de 1999, 724-732.

## ANEXO 1 – CÓDIGO VHDL

```
-- Definicao da entidade XPTO

Entity XPTO is
port (Clock: in bit;
        Reset: in bit;
        x: in bit;
        y: out bit);
end;

-- definicao da arquitectura XPTO_arch

architecture XPTO_arch of XPTO is
type Estados is (S1, S2);         --linha 11
signal Estado_presente: Estados;

begin
XPTO_machine: process (Clock, reset)
variable Count: INTEGER range 0 to 20;
begin                             --linha 17
if Reset='1' then                 --linha 18
    Count:=0;
    Count:=Count+1;
    y<='0';
    Estado_presente <= S1;        -- linha 22
elsif Clock'event and Clock = '1' then
    case Estado_presente is
        when S1 =>
            if Count<10 and x='0' then
                    Estado_presente <= S1;
                    Count:=Count+1;
                    y<='0';
            elsif Count=10 or x='1' then
                    Estado_presente <= S2;
                    y<='1';
            end if;
        when S2 =>
            if x='0' then
                    Estado_presente <= S1;
                    Count:=0;
                    Count:=Count+1;
                    y<='0';
            elsif x='1' then
                    Estado_presente <= S2;
                    y<='1';
            end if;
        when others =>
            null;
    end case;
end if;
end process;
end XPTO_arch;
```