CerVANTES

Co-VAlidatioN Tool for Embedded Systems

Autor: José Cabrita

Orientadores: José Costa e Paulo Flores

INESC-ID, grupo ALGOS

Índice

1 Objetivo

Este projeto tem como objetivo desenvolver um método de ligação para comunicação e transferência de dados entre dois simuladores. Um dos simuladores irá ser utilizado para efetuar a simulação do comportamento de componentes de *hardware* e outro simulador irá ser utilizado para efetuar a simulação da execução de um *software* num processador. As simulações deverão ocorrer independentemente com comunicação para transferência de dados entre os simuladores.

Para simular o *hardware* será utlizado o simulador de SystemC e para efetuar a simulação do comportamento do processador será utilizado um Instruction Set Simulator de maneira semelhante ao utilizado por Franco Fummi em [1].

2 Introdução

Hoje em dia um dos maiores desafios no desenvolvimento de sistemas é a integração de componentes de *hardware* e *software*. Geralmente são utilizadas ferramentas de co-simulação que permitem simular o comportamento do *hardware* utilizando ferramentas de simulação que utilizam uma dada descrição do *hardware* (*Hardware Description Language*) e o comportamento do *software* utilizando ferramentas de *debugg* regulares.

Geralmente as descrições de *hardware* são simuladas baseados em eventos ao passo que os simuladores de *software* são baseados em ciclos de relógio, este facto torna complicado obter simulações fiáveis quando existe interações entre *hardware* e *software*.

A co-simulação tem evoluído no sentido de conseguir obter descrições universais, ou seja, uma linguagem onde seja possível descrever *hardware* e *software* para que a simulação seja efetuada no mesmo simulador e se possa obter resultados fiáveis tanto do comportamento das partes independentes como do sistema completo com interações entre *hardware* e *software*.

A linguagem de descrição SystemC permite efetuar descrições de *hardware* e *software* pelo que consegue no seu simulador obter resultados de simulação de um sistema completo mesmo que utilize componentes de *hardware* e *software*. Uma das limitações desta simulação em SystemC é a necessidade de conhecimentos detalhados sobre o funcionamento do processador que vai executar o *software* a fim de se obter simulações corretas de acordo com o processador a ser utilizado.

Como sugerido por Franco Fummi em [1] vamos criar um modelo de interação entre o SystemC e um ISS que possa ser utilizado em qualquer ISS, pois geralmente quando um processador é desenvolvido é também desenvolvido um ISS próprio e assim é possível utilizar esse ISS em vez de o traduzir para a linguagem SystemC.

3 Modelo de Ligação SystemC-ISS

Em [1] Franco Fummi testa o funcionamento do simulador de SystemC com ligação a um simulador ISS utilizando 2 métodos diferentes. Em qualquer um destes métodos é assumido que o *hardware* é simulado em SystemC e o *software* simulado no ISS.

O primeiro método utiliza o simulador de SystemC para controlar toda a simulação. A comunicação entre o ISS e o SystemC é feita utilizando métodos de IPC (Inter-Process Comunication) onde é necessário criar elementos de SystemC que interagem com os elementos de comunicação IPC e é necessário também criar elementos no ISS que também interagem com os elementos de IPC.

Neste método são adicionados ao SystemC os elementos de comunicação para o ISS como variáveis do tipo iss_in e iss_out que apenas comunicam com o ISS, é também adicionado um processo específico para gerir a interação com o ISS (iss_process). Finalmente o gestor de eventos do simulador de SystemC é alterado para executar as funções de comunicação IPC presentes no iss_process sempre que exista alguma interação com elementos do tipo iss_in ou iss_out.

Do lado do ISS a comunicação IPC utiliza o *debugger* do ISS para ativar e desativar *breakpoints* sempre que seja necessário comunicar com o simulador de SystemC.

Neste caso os dois simuladores funcionam em simultâneo parando as suas simulações apenas quando existe comunicação entre os simuladores. Neste método a comunicação entre SystemC e ISS é completamente oculta do utilizador que tem apenas que desenvolver o *hardware* em SystemC e o *software* para o ISS, a comunicação ocorre quando existem alterações nas variáveis iss_in ou iss_out.

O segundo método utilizado por Franco Fummi descrito em [1] utiliza o ISS para controlar toda a simulação. Neste caso também é utilizado o SystemC exclusivamente para *hardware* e o ISS para *software*.

Neste caso a comunicação entre o ISS e os elementos de *hardware* em SystemC são efetuados recorrendo à criação de *drivers* de controlo de cada elemento de *hardware* e disponibilizando *sockets* para comunicação com os elementos de *hardware*. A comunicação é feita alterando o gestor de eventos do SystemC para receber mensagens do ISS que contem informação sobre a operação a realizar indicando se existe envio ou receção de dados.

Em [1] são explicadas as vantagens e desvantagens de cada um desses métodos e é também analisado um método de sincronização das simulações para que seja mais fácil depurar o sistema completo. Uma das soluções propostas é utilizar as comunicações para garantir que o simulador de SystemC e o ISS se encontram no mesmo ponto de simulação do sistema.

O método utilizado no projeto Cervantes teve em conta alguns aspetos apresentados nos dois métodos descritos. A comunicação utilizando drivers pode tornar complexo o desenvolvimento por parte do utilizador por ter de desenvolver um driver para cada elemento

de *hardware* em SystemC ao passo que na comunicação por IPC não será necessário desenvolver nenhum elemento de comunicação pois estarão já implementados nos elementos iss_in, iss_out e iss_process.

Assim é dada preferência à implementação do primeiro método recorrendo a comunicação IPC. Nesse método é também descrito que existe um controlo do *debugger* do ISS por parte do SystemC utilizando os elementos de comunicação IPC, o que torna as mensagens através dos canais IPC variadas e possivelmente com tamanhos também variados. No projeto Cervantes a comunicação IPC será utilizada unicamente como transferência de dados, assim garantimos que as mensagens são de um tamanho fixo e apenas enviam o número de bits necessário, para isso é necessário garantir do lado do ISS que a simulação aguarde quando existe transferência de dados entre os simuladores.

Da parte do simulador de SystemC no método descrito em [1] o gestor de eventos é alterado para verificar se existem interações com elementos iss_in e iss_out e quando existem interações então executar as funções relativas aos elementos de IPC. Neste caso pretende-se evitar a constante verificação de interações com os elementos de comunicação para evitar aumentar o número de operações no simulador de SystemC que pode ser significativo para o tempo real de simulação especialmente nos casos onde existam poucas interações entre os simuladores.

Tal como em [1] pretende-se implementar a comunicação ISS – SystemC com a possibilidade de sincronização de simulações para facilitar a depuração. Outra característica pretendida é a possibilidade de poder utilizar mais do que um simulador ISS para ser possível efetuar simulações multi-core.

Em suma o sistema a ser implementado deverá utilizar o simulador de SystemC para simular elementos de *hardware* e um ISS para elementos *software*. O simulador SystemC deverá controlar toda a simulação e comunicar através de IPC com o ISS. Deverá ser possível efetuar simulação multi-core utilizando um ISS para cada core e ter a possibilidade de efetuar sincronização da simulação do sistema geral através das comunicações IPC.

4 Implementação do sistema

A implementação do sistema tem como objetivo a utilização de qualquer ISS desde que seja possível efetuar as alterações necessárias. Para isso foram examinados alguns ISS comerciais (por exemplo Sourcery CodeBench da Menthor Graphics). No entanto os ISS verificados não disponibilizam o código fonte e os mecanismos de comunicação são de difícil adaptação a comunicação por IPC, assim foi utilizado um ISS genérico com o código fonte disponibilizado para que seja possível efetuar as alterações necessárias para o sistema de simulação proposto. Neste caso o ISS selecionado foi 8085 Instruction Set Simulator [2].

Para implementar a comunicação entre o ISS e o simulador de SystemC irá ser utilizado um elemento de IPC chamado *Pipe* utilizado em programação de redes em UNIX [3] cuja versão utilizada no sistema operativa Windows está disponível no *site* MSDN- the Microsoft

Developers Network [4]. Estes elementos permitem comunicação unidirecional ou bidirecional entre dois processos que se liguem ao *Pipe* onde um ficheiro virtual funciona como *buffer* de dados que irá conter a informação das mensagens a transmitir. Para criar o *Pipe* é necessário que um dos processos funcione como servidor que terá permissões para criar e eliminar o *Pipe*. O outro processo será o cliente do *Pipe* e ambos podem ter permissão para escrever e ler mensagens no *Pipe*.

Uma das características das comunicações utilizando um *Pipe* é a possibilidade dos processos que se liguem ao *Pipe* possam aguardar que a ligação ao outro processo seja efetuada, possibilitando assim uma garantia de comunicação entre os processos. Para isso é necessário que cada processo envie uma mensagem para o *Pipe* e fique à espera da resposta do outro processo. Por uma questão de conveniência o processo servidor envia uma mensagem para o *Pipe* e espera a resposta do cliente, do lado do cliente aguarda a mensagem enviada pelo servidor e envia a resposta.

Para o caso do sistema a implementar este método garante que os simuladores aguardem até que seja estabelecida a ligação entre os processos e as mensagens sejam trocadas. A utilização deste método permite que não seja necessário utilizar *breakpoints* na simulação do ISS (como é feito por Franco Fummi em [1]) desde que quando existe uma transferência de dados o processo do ISS aguarde que o processo do SystemC estabeleça a ligação do *Pipe*.

Esta característica é também útil na questão da sincronização das simulações pois pode ser efetuado um pedido de transferência de dados sem que realmente existam dados a ser transferidos, garantindo assim que os simuladores aguardam num dado ponto da simulação do sistema completo e apenas continuam as respetivas simulações quando os dois simuladores se encontram nesse ponto.

Existe também a possibilidade de vários processos clientes utilizarem o mesmo *Pipe* para comunicar com o processo servidor, esta característica é útil para a questão de simulação multi-core, no entanto, a presença de vários processos cliente leva a que o numero de mensagens disponível no *Pipe* possa estar desorganizada pelo que seria necessário enviar mais dados nas mensagens e não só dados a transferir. É também possível, um servidor de Pipes abrir um certo número de *Pipes* diferentes, este número vai estar limitado de acordo com os recursos disponíveis no sistema, geralmente, para os sistemas atuais, este número estará na casa dos milhares. Assim é possível, utilizando o simulador de SystemC como servidor, criar um *Pipe* para cada processo ISS criado e assim poder enviar apenas os dados a transferir na mensagem para o *Pipe* tal como pretendido.

4.1 Alterações no SystemC

No sistema a implementar escolhemos o simulador do SystemC para servir como servidor dos elementos de comunicação IPC e assim controlar a simulação do sistema completo.

Considerando o objetivo estabelecido para a comunicação IPC no simulador, foram criadas um conjunto de funções para efetuar o controlo dos elementos IPC no simulador SystemC. Estas funções foram acrescentadas à biblioteca de SystemC (SystemC.lib) através dos ficheiros sc_IPC.h e sc_IPC.cpp.

As funções para controlar os elementos IPC que são utilizadas, são as referidas em [4] para utilização em Windows, a utilização destes elementos de comunicação noutro Sistema Operativo deverão ser substituídas pelas funções respetivas, pois o funcionamento é semelhante noutros sistemas operativos e geralmente baseados nas descrições utilizadas em UNIX [3].

Nos ficheiros de IPC foram criadas 6 funções, “void open_iss()”, “HANDLE create_pipe()”, “HANDLE create_pipe(int ID)”, ”BOOL wait_client_connecct(HANDLE wpipe)”, “int pipe_data_exchange(HANDLE hPipe, int data)” e “void close_pipe_connection(HANDLE nPipe)”.

A função “void open_iss()” é utilizada para criar um processo para o ISS, este processo utiliza o executável do ISS, neste caso o “8085.exe”. Cada vez que é chamada esta função é criado um novo processo independente dos processos ISS já existentes.

A função “HANDLE create_pipe()” é utilizada para criar um *Pipe* onde o SystemC é o servidor. O *Pipe* criado por esta função é utilizado como elemento de comunicação para enviar um valor que representa a identidade (ID) do ISS aberto, este valor irá indicar qual o *Pipe* a que cada processo ISS deverá efetuar a ligação para enviar e receber dados, para isso cada ISS tem um ID específico. O *Pipe* criado nesta função é utilizado exclusivamente na comunicação do ID para os processos de ISS. O valor “HANDLE” retornado pela função é um elemento de controlo que aponta para o *Pipe* aberto pela função.

A função “HANDLE create_pipe(intID)” é utilizada para criar um *Pipe* de comunicação de dados para um dado ISS. De acordo com o valor ID é utilizado um *Pipe* diferente que serve de ligação para um ISS diferente de acordo com o ID. Esta função funciona de maneira semelhante a “HANDLE create_pipe()” e também retorna o valor “HANDLE” respetivo ao *Pipe* aberto.

A função “BOOL wait_client_connection(HANDLE wpipe)” é utilizada para o servidor de *Pipes* verificar se o processo cliente (neste caso o ISS) já está ligado ao *Pipe*. Para garantir a ligação dos dois elementos, e assim garantir a sincronização das simulações, esta função aguarda que o processo cliente esteja ligado ao *Pipe* e retorna um valor “TRUE” se estabelecer ligação e ”FALSE” se houver algum problema ao tentar estabelecer a ligação.

A função “int pipe_data_exchange(HANDLE hPipe, int data)” inclui a troca de dados entre os simuladores ISS e SystemC para um dado *Pipe*. Esta função recebe um valor “int data” para

enviar para o ISS e retorna um valor “int” com o valor que foi recebido do ISS. Sempre que é estabelecida uma ligação para troca de dados entre o ISS e o SystemC é sempre enviada uma mensagem do SystemC e recebida uma mensagem do ISS, dependendo dos elementos que fazem a chamada a esta função os valores validos vão ser diferentes, ou seja, se o SystemC receber dados do ISS o valor “int data” tem um valor genérico e o valor válido é o valor “int” retornado. No caso de o SystemC enviar dados para o ISS o valor “int data” é válido e o valor “int” retornado é um valor genérico.

Finalmente a função “void close_pipe_connection(HANDLE nPipe)” é utilizada para fechar a ligação do servidor a um dado *Pipe*. É necessário fechar a ligação ao *Pipe*, pois para manter a ordem correta da simulação do sistema completo apenas o *Pipe* utilizado na transferência que está a decorrer deverá estar aberto para que outros processos ISS aguardem a abertura do respetivo *Pipe*.

Para que o utilizador possa definir se pretende utilizar a simulação com ISS e quantos processos ISS quer utilizar, foi criada a função “sc_ISS(int num_instances)” e acrescentada à biblioteca SystemC (Systemc.lib) no ficheiro “sc_simcontext.cpp”. Esta função (Figura 1) é utilizada para criar os processos ISS de acordo com o valor em “int num_instances” e enviar a cada processo ISS o respetivo ID para que o processo ISS saiba qual o *Pipe* a que corresponde. Para utilizar a simulação com ISS o utilizador deverá incluir no seu sc_main, e antes da função de chamada de simulação “sc_start”, a função “sc_ISS(x)”, onde x representa o número de processos ISS, apenas uma vez.

```
void
sc_ISS( int num_instances){

    int ID=1;
    HANDLE pipe;


    while(ID<=num_instances){
        open_iss();
        pipe=create_pipe();
        wait_client_connect(pipe);
        pipe_data_exchange(pipe,ID);
        close_pipe_connection(pipe);
        ID++;
    }

    return;
}
```

Figura 1 - Função "sc_ISS(int num_instances)"

.

Para evitar as alterações ao *kernel* de SystemC feitas por Franco Fummi [1], optou-se pela inserção de funções de comunicação e controlo dos elementos de IPC nos elementos SystemC. Para isso foram criados dois novos elementos SystemC, um sc_iss_in e um sc_iss_out e acrescentados à classe base de elementos de SystemC sc_core. Estes elementos partilham de todas as características dos elementos sc_in e sc_out respetivamente e podem também ser utilizados como sc_in e sc_out.

Os elementos sc_in e sc_out utilizam funções de obtenção e armazenamento de elementos para a simulação. Para um porto genérico a obtenção do valor é feita utilizando porto.read() e o armazenamento é feito com porto.write(variável). Estas funções são utilizadas como base para as funções de comunicação IPC e forma acrescentadas nos ficheiros sc_signal.h e sc_signal.cpp.

No caso do elemento sc_iss_in, que recebe dados do ISS, é criada a função "issread(x)" (Figura 2) utilizada como "porto.issread(x)" (Figura 3, mostra o valor recebido da função porto.issread(x) a ser transportado para uma variável temporária SystemC e escrito num porto de saída de um modulo) onde x representa a identificação do processo ISS. Esta função utiliza as funções de comunicação IPC para comunicar com o processo ISS selecionado e receber o valor enviado pelo ISS. O valor de comunicação enviado para o ISS não é importante pois do lado do ISS não vai ser considerado. É aconselhável armazenar o valor recebido numa variável temporária de SystemC pois a função issread(x) não converte o valor recebido para o tipo de variáveis utilizadas na simulação SystemC e assim a passagem do valor retornado para uma variável temporária SystemC assegura que o valor é convertido.

```
virtual const T& issread(int ID) const
{
    int out_data = 0, in_data=0;
    HANDLE pipe;

    pipe=create_pipe(ID);
    wait_client_connect(pipe);
    in_data=pipe_data_exchange(pipe, out_data);
    close_pipe_connection(pipe);

    return in_data;}
```

Figura 2 - Função issread(x)

```
if(receive_from_iss1 == 1){
    temp = stub_for_data_from_iss.issread(1);
    data_from_iss.write(temp);
}
```

Figura 3 - Exemplo de utilização da função porto.issread(x).

Para enviar dados para o ISS é utilizado o elemento sc_iss_out onde foi criada a função "isswrite(data,x)" (Figura 4) utilizada como "porto.wrtite(data,x)" (Figura 5) onde data é o valor a ser transmitido para o ISS e x a identificação do processo do ISS. Tal como a função "issread(x)" esta função utiliza as mesmas funções de comunicação IPC para comunicar com o processo de ISS indicado, mas trata os dados de maneira diferente pois o que interessa agora é o valor enviado ao passo que o valor recebido não é considerado.

```
inline
void
sc_signal<bool>::isswrite( const bool& value_ , int ID)
{
    HANDLE pipe;
    int dados;

    pipe = create_pipe(ID);

    wait_client_connect(pipe);

    dados = pipe_data_exchange(pipe, value_ );

    close_pipe_connection(pipe);
}
```

Figura 4 - Função isswrite(data,x).

```
if(send_to_iss1 == 1){
    temp = data_for_iss.read();
    stub_for_data_for_iss.isswrite(temp,1);
}
```

Figura 5 - Exemplo de utilização da função porto.isswrite(data,x).

A utilização destas funções só está disponível nos portos do tipo sc_iss_in e sc_iss_out e o utilizador deverá garantir que estas funções são chamadas apenas durante um ciclo de relógio da simulação, pois cada vez que são chamadas efetuam uma comunicação com o ISS. Na Figura 6 e Figura 7 temos um exemplo de um módulo utilizado para comunicar com dois processos de ISS onde podemos verificar a definição dos portos de comunicação com o ISS e a utilização de variáveis de controlo para garantir que as funções são chamadas apenas uma vez. De salientar que a pré simulação do SystemC executa as funções dos módulos uma vez antes de iniciar a simulação o que pode levar a pedidos de comunicação se as funções "issread(x)" ou "isswrite(data,x)" forem utilizadas sem controlo.

```
SC_MODULE (ISS_CONNECT){

    // input ports
    sc_in<bool> send_to_iss1; // high envia dados para iss
    sc_in<bool> send_to_iss2; // high envia dados para iss
    sc_in<bool> receive_from_iss1; // high recebe dados do iss
    sc_in<bool> receive_from_iss2; // high recebe dados do iss
    sc_iss_in<sc_uint<8>> stub_for_data_from_iss; //ligaçao para receber dados do iss
    sc_in<sc_uint<8>> data_for_iss; // envio de dados para o iss

    // output ports
    sc_out<sc_uint<8>> data_from_iss; // recebimento de dados do iss
    sc_iss_out<sc_uint<8>> stub_for_data_for_iss; //ligaçao para enviar dados para o iss

    //local variables
    sc_uint<8> temp;

    // CONSTRUCTOR
    SC_CTOR(ISS_CONNECT){
        SC_THREAD(iss_com);
        sensitive << send_to_iss1;
        sensitive << send_to_iss2;
        sensitive << receive_from_iss1;
        sensitive << receive_from_iss2;
    }
```

Figura 6 - Exemplo de definição de portos, sinais e lista de sensibilidade do módulo de comunicação com ISS

```
    //module function
    void iss_com(){
        while(1){
            if(send_to_iss1 == 1){
                temp = data_for_iss.read();
                stub_for_data_for_iss.isswrite(temp,1);
            }

            if(receive_from_iss1 == 1){
                temp = stub_for_data_from_iss.issread(1);
                data_from_iss.write(temp);
            }

            if(send_to_iss2 == 1){
                temp = data_for_iss.read();
                stub_for_data_for_iss.isswrite(temp,2);
            }

            if(receive_from_iss2 == 1){
                temp = stub_for_data_from_iss.issread(2);
                data_from_iss.write(temp);
            }
            wait();
        }
    }
};
```

Figura 7 - Exemplo da função do módulo de comunicação com ISS.

Na Figura 8 temos o exemplo de definição do módulo de comunicação com o ISS onde podemos verificar que é necessário ligar um sinal aos portos de comunicação com o ISS apesar de não ser utilizado o valor transmitido para esses sinais. A necessidade de manter estas ligações não foi retirada para manter a estabilidade da simulação SystemC.

```
sc_signal<bool> send_to_iss1;
sc_signal<bool> send_to_iss2;
sc_signal<bool> receive_from_iss1;
sc_signal<bool> receive_from_iss2;
sc_signal<sc_uint<8>> stub_for_data_from_iss;
sc_signal<sc_uint<8>> data_for_iss;
sc_signal<sc_uint<8>> data_from_iss;
sc_signal<sc_uint<8>> stub_for_data_for_iss;


//modulo de comunicação com iss
ISS_CONNECT ISS("ISS");
    ISS.send_to_iss1(send_to_iss1);
    ISS.send_to_iss2(send_to_iss2);
    ISS.receive_from_iss1(receive_from_iss1);
    ISS.receive_from_iss2(receive_from_iss2);
    ISS.stub_for_data_from_iss(stub_for_data_from_iss);
    ISS.data_for_iss(data_for_iss);
    ISS.data_from_iss(data_from_iss);
    ISS.stub_for_data_for_iss(stub_for_data_for_iss);
```

Figura 8 - Definição do módulo de comunicação com ISS no sc_main().

O tamanho dos dados transferidos para o ISS está limitado a 8 bits no caso deste exemplo, pois o ISS do 8085 utiliza registo de 8 bits e para manter a lógica da simulação não se justifica enviar mais que 8 bits numa transferência de dados. No caso de ser utilizado um ISS com uma maior capacidade para transferência de dados é apenas necessário aumentar o número de bytes transferidos nas funções de comunicação IPC, neste caso na compilação da biblioteca SystemC no ficheiro “SC_IPC.cpp”, na função “int pipe_data_exchange(HANDLE hPipe, int Data)” alterar o valor de “cbReplyBytes” para o número de bytes desejados para o envio de dados. Do lado do ISS será necessário também alterar o número de bytes enviados.

4.2 8085 Instruction Set Simulator (ISS)

O ISS escolhido é um simulador bastante simples e apenas permite a simulação de código escrito em *assembler* utilizando apenas operações do conjunto de operações disponíveis no microprocessador de 8 bits 8085 da Intel. O simulador utiliza um *debugger* próprio e permite efetuar simulações passo a passo e verificação de valores nos registos e memoria de microprocessador em qualquer ponto da simulação.

O processador 8085 tem um conjunto de 74 operações diferentes disponíveis na documentação em [2] e utiliza um conjunto de 7 registos de 8 bits acessíveis para programação onde um dos registos funciona como acumulador para entrada da unidade de cálculos (ALU – Arithmetic Logic Unit). O simulador utiliza várias combinações das 74 operações totalizando um total de 256 operações disponíveis para a simulação do processador (a operação de movimento de valores de valores entre registos é contabilizada utilizando todas as combinações possíveis de troca de valores entre registos).

O simulador utiliza linguagem C++ para implementação dos elementos do simulador e Java para a interface gráfica. O simulador utiliza os mecanismos disponíveis do Sistema Operativo Windows para todas as operações de interação (abrir ficheiros, botões de interface gráfica etc…).

Na Figura 9 temos um exemplo de simulação a mostrar os valores nos registos e a lista de operações.

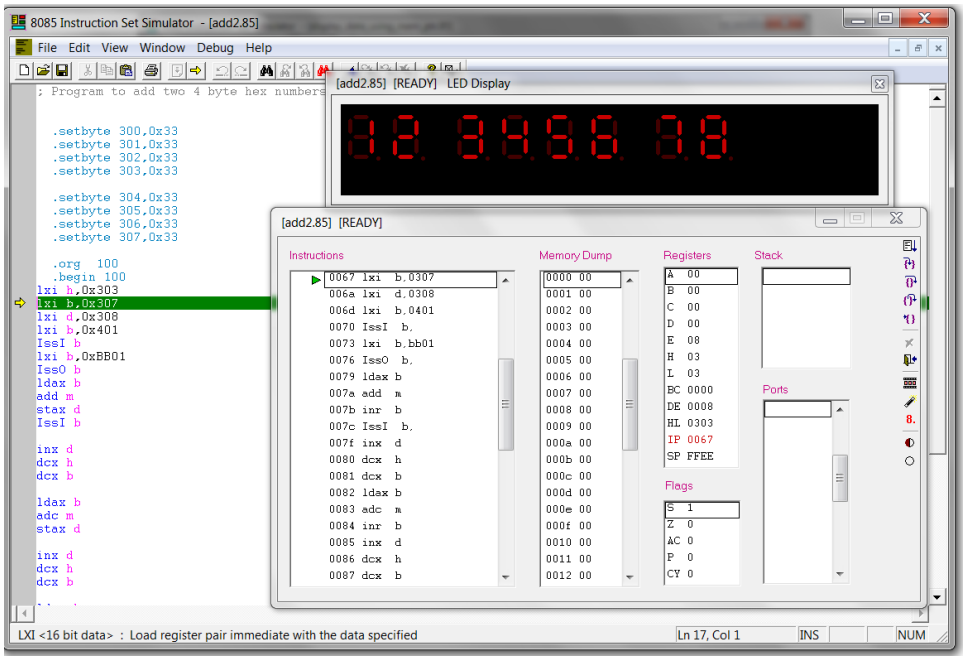

Figura 9 – Exemplo do ambiente de simulação do 8075.

4.3 Alterações no ISS

De modo a acomodar a comunicação IPC no ISS do 8085 sem alterar o simulador foram acrescentadas 2 operações à lista de operações simuladas de maneira a simular uma comunicação externa de um periférico que envia dados para o processador 8085. Estas operações efetuam movimentos de dados para um dos registos do processador, neste caso o registo escolhido foi o B. Quando existe uma transferência de dados do ISS para o SystemC é sempre o valor presente no registo B que é transferido, mas não é alterado. E no caso de uma transferência de dados do SystemC para o ISS o valor do registo B é perdido para acomodar os dados recebidos.

O ISS é sempre iniciado através de um processo criado pelo SystemC e como cada processo tem um *Pipe* de IPC próprio é necessário que o ISS saiba qual o seu ID. Para isso foi criada a função "void get_instance()" que executa sempre que o ISS é iniciado. Esta função efetua a ligação ao *Pipe* criado pelo SystemC na função "void create_pipe()". Quando a ligação fica estabelecida envia uma mensagem de reconhecimento e na resposta recebe o ID enviado pelo SystemC. O valor do ID é guardado numa variável global definida no ficheiro "8085Doc.h" para ser utilizada para ligar ao *Pipe* correto. A função "void get_instance()" apenas é executada uma vez.

O simulador de ISS do 8085 tem um conjunto de 256 operações e utiliza múltiplas verificações para manter a estabilidade da simulação. De modo a não efetuar alterações aos mecanismos de estabilidade optou-se por substituir 2 operações da lista de 256 para incluir as funções de comunicação. No ficheiro "InstructionSet.cpp" foram removidas as operações "mov b,b" e "mov a,a" (operações que não são utilizadas, transferência do valor do registo para o próprio registo) para acomodar as operações "IssI b" e "IssO b" para receber dados do SystemC e enviar dados para o SystemC respetivamente.

As funções que são executadas quando se utiliza "IssI b" e "IssO b" estão definidas no ficheiro "8085Object.cpp" como "int C8085Object::VKIssI()" e "int C8085Object::VKIssO()" respetivamente.

O funcionamento das funções é semelhante na parte da comunicação IPC, ou seja, em ambas as funções o processo é verificar se o *Pipe* existe e aguardar até que exista, efetuar a ligação ao *Pipe*, enviar uma mensagem e aguardar a resposta. No caso da função "IssO b" a mensagem enviada contem os dados a ser enviados presentes no registo B do processador 8085 e a mensagem recebida é ignorada. No caso da função "IssI b" a mensagem enviada é genérica e a mensagem recebida tem os dados transferidos e é armazenada no registo B.

De salientar que quando uma destas funções é executada, o simulador aguarda sempre que exista uma ligação no *Pipe* evitando assim a necessidade de manipular *breakpoints* no ISS como feito por Franco Fummi em [1].

O recebimento de dados do SystemC está limitado a 8 bits pois o registo B é um registo de 8 bits, se estas funções forem utilizadas num outro ISS com capacidade para transferências

de dados maiores basta alterar o valor “cbToWrite” para o número de bytes desejado na função “int C8085Object::VKIssO()” no ficheiro “8085Object.cpp”.

5 Resultados

O módulo de exemplo descrito em Figura 6, Figura 7 e Figura 8 foi utilizado para testar a co-simulação do SystemC com o ISS do processador 8085 utilizando 2 processos de ISS.

A simulação inicia-se com a utilização da função “sc_iss(2)” para iniciar 2 processos de ISS com os IDs 1 e 2 respetivamente. Quando os simuladores ISS já tiverem iniciado a interface gráfica indica que já estão prontos para escrever ou abrir um ficheiro de código *assembler* para simulação. Após abertura do ficheiro pode-se fazer a simulação *step-by-step* utilizando o *debugger* do ISS ou simplesmente fazer “*run*” do código para efetuar a simulação. Neste ponto o simulador do SystemC já deverá estar à espera da primeira troca de comunicação.

É então enviado um valor para o ISS1 e outro para o ISS2 e nos ISSs os valores recebidos são alterados e reenviados para o SystemC. Foi também necessário garantir que as funções de envio e receção de valores no SystemC estão ativas apenas num ciclo de relógio de simulação através dos sinais de controlo e da utilização da função “sc_start(1, SC_NS)”. Na Figura 10 temos o exemplo da bancada de teste criada com o módulo de teste.

```
data_for_iss = 0;send_to_iss1 = 0;send_to_iss2 = 0;receive_from_iss1 = 0;receive_from_iss2 = 0;

sc_ISS(2);
sc_start(10,SC_NS);

data_for_iss = 55;send_to_iss1 = 1;send_to_iss2 = 0;receive_from_iss1 = 0;receive_from_iss2 = 0;

sc_start(1,SC_NS);

data_for_iss = 66;send_to_iss1 = 0;send_to_iss2 = 1;receive_from_iss1 = 0;receive_from_iss2 = 0;

sc_start(1,SC_NS);

data_for_iss = 66;send_to_iss1 = 0;send_to_iss2 = 0;receive_from_iss1 = 1;receive_from_iss2 = 0;

sc_start(1,SC_NS);

cout << "dados recebidos de 1 = " << data_from_iss << endl;

data_for_iss = 66;send_to_iss1 = 0;send_to_iss2 = 0;receive_from_iss1 = 0;receive_from_iss2 = 1;

sc_start(1,SC_NS);

cout << "dados recebidos de 2 = " << data_from_iss << endl;
```

Figura 10 - Bancada de teste do módulo de comunicação com ISS.

A simulação comportou-se como o esperado e tanto os valores enviados como os recebidos nas comunicações com os ISS estavam corretos de acordo com o esperado.

6 Conclusões

Dos elementos de comunicação analisados verificou-se que a utilização de *Pipes* para as transferências de dados seria o mais adequado. Esta opção tornou-se mais obvia quando se confirmou a possibilidade de parar os simuladores utilizando apenas os mecanismo de comunicação IPC eliminando assim a necessidade de existir alguma troca de dados de controlo entre o SystemC e o ISS limitando as transferências aos valores de simulação a transmitir.

Dependendo do ISS final a utilizar poderá recorre-se a uma opção que não foi mencionada. Neste caso o *Pipe* de comunicação é sempre aberto para uma transferência e fechado mesmo que mais tarde volte a ser aberto. Para o caso de a abertura e fecho dos *Pipes* provocar uma limitação no tempo de simulação é possível manter um *Pipe* aberto utilizando uma Classe especifica que efetue a gestão da comunicação do *Pipe*, tanto no simulador de SystemC como no ISS e assim permita que o *Pipe* esteja sempre aberto. De salientar que esta solução poderá exigir muitos recursos do PC a ser utilizado pois terá tantos *Pipes* abertos como processos de ISS iniciados.

A utilização de mecanismos de controlo de transferências que estão mais visíveis para o utilizador (comparando com o que foi feito por Franco Fummi em [1]) pode levar a que seja mais complexo efetuar um projeto de co-simulação, no entanto poderá também ser útil em alguns casos a perceção do mecanismo de comunicação.

Os objetivos propostos foram atingidos, no entanto o sistema final carece de alguns testes nomeadamente no campo da performance temporal das simulações que poderá levar a algumas alterações à versão final.

7 Referências

[1] F. Fummi, S. Martini, G. Perbellini e M. Poncino, “"Native ISS-SystemC Integration for the Co-Simulation of Multi-Processor SoC",” em *Conference on Design, automation and test*, Europe, 2004.

[2] “8085 Microprocessor simulator,” [Online]. Available: http://8085.codeplex.com/.

[3] W. R. Stevens, UNIX NETWORK PROGRAMMING Interprocess Communications Volume 1 & 2, Prentice Hall, Inc, 1999.

[4] “MSDN - The Microsoft Developer Network (Named Pipes),” Microsoft, [Online]. Available: http://msdn.microsoft.com/en-us/library/windows/desktop/aa365590(v=vs.85).aspx.