Help on the Embedded Software Block

Powersim Inc.

1. Introduction

The Embedded Software Block is a block that allows users to model embedded devices such as microcontrollers, DSP, or other devices. It is a variation of the general DLL block. For more information on the use of the general DLL block, please refer to the document "Help General DLL Block.pdf".

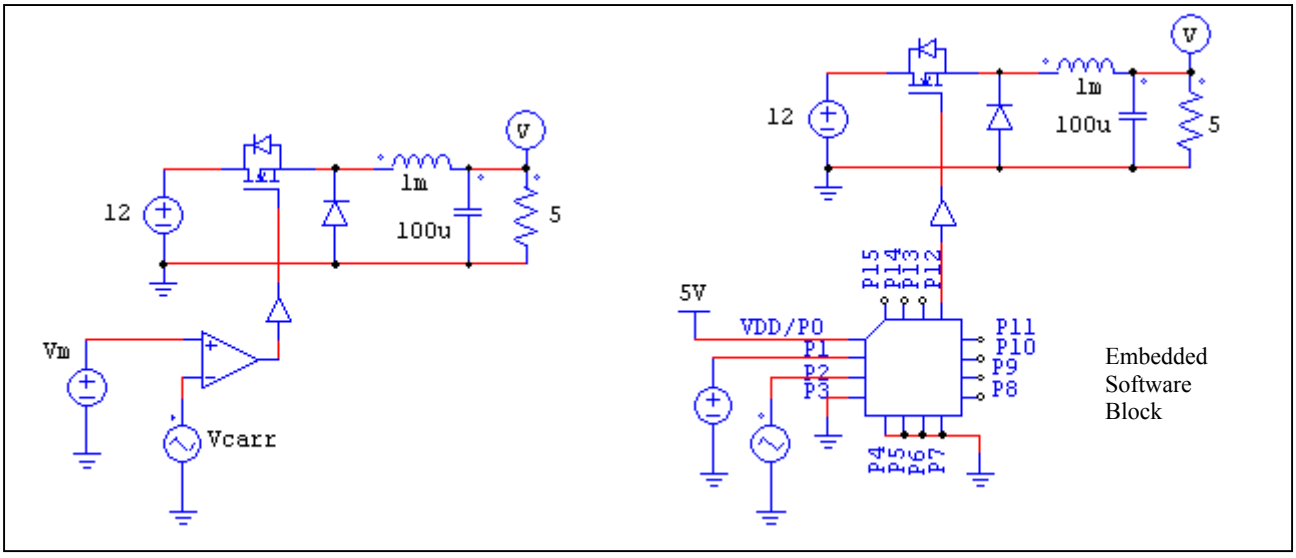
However, unlike the general DLL block whose connection nodes are fixed and are predefined as either inputs or outputs, the node types of the Embedded Software Block can be programmed as needed.

Also, additional information, such as the exact instant at which the state of a variable changes, can be calculated and passed to PSIM. This information can be used to minimize errors introduced by fixed time step simulation.

2. How to Use the Embedded Software Block

We will use a comparator example to illustrate how to use the Embedded Software Block. For more details and the complete source code, one should refer to the example (schematic file: "comparator_ESB.sch"; source code: "comparator.cpp")

The circuit on the left shows a buck converter with the gating signal generated by a comparator. The circuit on the right shows the same circuit, except that the function of the comparator is implemented using an Embedded Software Block.



In this example, the Embedded Software Block has 16 ports. Ports P0 through P7 are inputs, and Ports P8 through P15 are outputs.

The properties of the block, including the number of ports, port input/output types, port names, input parameters, are defined in the custom C++ code. The code includes the following data structure and functions:

struct Internal_DLL_Block_SimulationData:	Variables used internally by DLL are defined here.
void REQUESTUSERDATA:	Function that defines port number, names, and parameters.
void OPENSIMUSER:	Function that reads parameters from the interface and performs initialization.
void STARTSIMUSER:	Function that defines the port type (input or output).
void RUNSIMUSER2:	Function that reads inputs from PSIM, performs calculation, and sends results back to PSIM.

Defining Port Number and Parameters:

The first step to define the Embedded Software Block is to define the number of ports and parameters. They are defined in the function **REQUESTUSERDATA**, as shown in the code below, highlighted in red:

```
void REQUESTUSERDATA(int nRequestReason, int nRequestCode, int nRequestParam, void ** ptrUserData, int * pnParam1,
int * pnParam2, char * szParam1, char * szParam2)
{
… … … … …
switch(   nRequestReason )
{
        case ACTION_DLL_SELECTED:
        {
                switch(nRequestCode)
                {
                        … … … …
                        case REQUEST_IN_OUT_NODES:
                                *pnParam1 = 16;                 //Define 16 ports
                                *pnParam2 = 0;
                                return;
                        case REQUEST_INPUT_NODE_INFO:           //Define port names
                                nNode = nRequestParam;
                                switch(nNode)
                                {
                                        case 0:
                                                strcpy(szParam1, "VDD/P0");
                                                break;
                                        case 1:
                                                strcpy(szParam1, "P1");
                                                break;
                                                … … … …
                                }
                                return;
                        case REQUEST_PARAM_COUNT:
                                *pnParam1 = 1;                  //Define 1 parameter
                                *pnParam2 = 0;
                                strcpy(szParam1, "All Files|*.*||"); //File Open Dialog Filter for InputFile.
                                return;
                        … … … …
                        case REQUEST_PARAM_INFO:
                        {
                                switch(nRequestParam)
                                {
                                        case 0:                 //Define parameter name
                                        strcpy(szParam1, "Flag for Exact Switching");
                                        strcpy(szParam2, "1");  //Set default parameter value as 1
                                        *pnParam1 = 1;          //Show Display check box
                                        break;
                                        … … … …
                                }
                        … … … …
```

Reading Parameters from the Interface:

The next step is to read parameters from the interface. In this case, one parameter, "Flag for exact switching", is defined. This parameter will appear in the dialog window of the Embedded Software Block. To read the parameter value into the DLL, first define an internal variable called "flag_exact_switching" in the DLL structure **Internal_DLL_Block_SimulationData**. Then read the parameter value into DLL as shown in the code below.

```
void OPENSIMUSER(const char *szId, const char * szNetlist, void ** ptrUserData, int *pnError, LPSTR szErrorMsg, void *
pPsimParams)
{
        EXT_FUNC_PSIM_INFO * pPsimInfo = (EXT_FUNC_PSIM_INFO *)pPsimParams;
        assert(*ptrUserData == NULL);
        *ptrUserData = new Internal_DLL_Block_SimulationData;
        Internal_DLL_Block_SimulationData * pData = (Internal_DLL_Block_SimulationData *)(*ptrUserData);
        memset(pData, 0, sizeof(Internal_DLL_Block_SimulationData) );

        … … … …
        pData->m_nInputNodes  = atoi(netlist[2]);
        pData->m_nOutputNodes = atoi(netlist[3]);
        int nParamStartIndex = 5 + pData->m_nInputNodes + pData->m_nOutputNodes;

        // Read the value of Parameter 1 from the interface
        pData->flag_exact_switching = atoi( netlist[nParamStartIndex] );

        //Initialize internal DLL data
        pData->Vgat0 = 0.;

        *pnError = 0; //Success
}
```

In the code, pData is the pointer to the internal DLL data structure, and both "flag_exact_switching" and "Vgat0" are variables defined in the structure. Also, any initialization of the internal variables, such as the initialization of the variable Vgat0, is performed here.

Defining the Port Type:

The port types of an Embedded Software Block need to be defined in the code. A port can be defined as either input or output. In this example, Ports P0 through P7 are the inputs, and Ports P8 through P15 are the outputs. They are defined as shown in the code below.

```
void STARTSIMUSER(int *portTypes, void ** ptrUserData, int *pnError, LPSTR szErrorMsg)
{
        Internal_DLL_Block_SimulationData * pData = (Internal_DLL_Block_SimulationData *)(*ptrUserData);
        if( pData == NULL) { return; }

        for (int i=0; i<8; i++)
        {
                portTypes[i] = TYPE_PORT_INPUT;          //Define P0 through P7 as inputs
        }

        for (i=8; i<16; i++)
        {
                portTypes[i] = TYPE_PORT_INPUT;          //Define P8 through P15 as outputs
        }

        *pnError = 0; //Success
}
```

Implementing the Function of a Comparator:

The function of the comparator is implemented in the routine **RUNSIMUSER2** which is called at every time step. Two arrays, *ports* and *ports2*, are used to transfer data between PSIM and the DLL. The array *ports* stores the voltages at each port. For example, in this example, *ports[1]* and *ports[2]* store the non-inverting input (Port P1) and the inverting input (Port P2) of the comparator, and *ports[12]* stores the output of the comparator (Port P12).

The implementation in the code is shown below.

```
void RUNSIMUSER2 (double t, double delt, double *ports, double *ports2, int *portTypes, void ** ptrUserData, int *pnError,
LPSTR szErrorMsg)
{
        Internal_DLL_Block_SimulationData * pData = (Internal_DLL_Block_SimulationData *)(*ptrUserData);
        if( pData == NULL) { return; }

        double Vm, Vcarr, Vm0, Vcarr0, Vdd;
        int iflag;

        Vdd = ports[0];                                         //Inputs
        Vm = ports[1];
        Vcarr = ports[2];
        iflag = 0;                                              //If iflag=1, output has changed the state.

        if (Vm >= Vcarr)                                        //Implement the function of a comparator
        {
                ports[12] = 1.;
                if (pData->Vgat0 < 0.5) iflag = 1;
        }
        else
        {
                ports[12] = 0.;
                if (pData->Vgat0 > 0.5) iflag = 1;
        }

        if (iflag && pData->flag_exact_switching)               //Calculate the exact switching instant
        {
                Vm0 = pData->Vm0;
                Vcarr0 = pData->Vcarr0;
                ports2[12] = delt*(Vm-Vcarr)/(Vm-Vm0-Vcarr+Vcarr0);
        }

        pData->Vm0 = ports[1];                                  // Store the value for use in the next step
        pData->Vcarr0 = ports[2];
        pData->Vgat0 = ports[12];

        *pnError = 0; //Success
}
```

The other array, *ports2*, is used to store the information of the exact instant of the change of a state. Since PSIM uses fixed time step simulation, a variable may change the state (for example, the output of a comparator or a logic gate changes from 0 to 1) between the two discrete points. For the sake of discussion, the time interval between the exact instant of state change and the current time is referred as *time error*. The array *ports2*, therefore, is used to store the time errors for each port. If the port is an input, the time error is passed from PSIM to DLL, and if the port is an output, the time error is passed from DLL to PSIM.

If the time error is not taken into account, a small error will be introduced into the simulation, especially if the time step is large. However, PSIM can eliminate the simulation error by taking the time error into account in the calculation.

In this example, when the comparator output changes the state, the code calculates the time error using the present and historic values of the non-inverting and inverting inputs. This value is then stored in the array *ports2* and passed to PSIM.

To see the effect of the error correction, one can set the parameter "flag for switching instant" to 0, and run the simulation. The part of the circuit that uses the comparator will have the time error correction, and the circuit that uses the Embedded Software Block will have no time error correction. One would notice a small difference between these two inductor currents.

Customizing the Image of the Embedded Software Block:

Once the code is compiled and the DLL file for the Embedded Software Block is created, the block can be used in a circuit schematic, and the image of the block can be customized. Follow the procedure below to load the block and customize the image:

- In PSIM, go to **Elements** -> **Control** -> **Other Function Blocks**. Select **Embedded Software Block**, and place it on the schematic.
- Double click on the block, and click on the browser button at the end of the input field **DLL File**, and select the DLL file. In this example, the file "comparator_ESB.dll" will be selected.
- Once the DLL file is selected, the block image will be changed to a rectangle, with all the nodes arranged on the left, and node sequence listed from the top to the bottom.
- To change the size of block, double click on the block, and click on the **Edit Image** button to enter into the image editor. Delete the pre-drawn rectangle and the arrow lines on the left. Then go to **File** -> **Set Image Size** to set the width and height of the block. In this example, both the width and height are set to 7.
- To rearrange the location of the ports, first clear all the ports by going to **File** -> **Reset Outputs**. Then left click on the diamonds to define the port location. In this example, for instance, click on the 2nd diamond from the top on the left for the location of Port P0, and a number 1 will appear inside the diamond which represent the first port.
- Redraw the block image using the drawing utility.