Implementation of a SM drive in a voltage-source converter control system with a PSCad/EMTDC simulation software interface

Frank Johansson

May 20, 2002
The main goal of the Platform Pre-compression project is feeding a certain mechanical torque into a gas compressor at a certain shaft speed. This is accomplished by driving a Motorformer™ with a high voltage force-commutated converter fed by two submarine cables connected to a rectifier on-shore.

A standard control system for the converter has been modified in order to accommodate the motor drive controller. The input and output signals are slightly different, there are new parameters required and there is a significant amount of control functions that are obsolete for the platform station. The simulation interface has been adapted to the new control system. The system needs further development, but the low-level software is fully operational.

Keywords:
Abstract

The main goal of the Platform Pre-compression project is feeding a certain mechanical torque into a gas compressor at a certain shaft speed. This is accomplished by driving a Motorformer™ with a high voltage force-commutated converter fed by two submarine cables connected to a rectifier on-shore.

A standard control system for converter has been modified in order to accommodate the motor drive controller. The input and output signals are slightly different, there are new parameters required and there is a significant amount of control functions that are obsolete for the platform station. The simulation interface has been adapted to the new control system. The system needs further development, but the low-level software is fully operational.
Table of contents

Abstract 1

Table of contents ................................................................. 2

1 Introduction ........................................................................ 3
  1.1 Goals ............................................................................. 3
  1.2 Description of application ............................................. 3
  1.3 Previous work ............................................................... 3

2 The HVDC Light™ control system........................................ 4
  2.1 The control hardware: Mach2 ....................................... 4
  2.2 The control software: C_system_hidraw ....................... 5
    2.2.1 Software functionality ........................................... 5
    2.2.2 Target platforms .................................................. 6
    2.2.3 Binary generation ................................................ 6

3 Simulation model ................................................................... 8
  3.1 The mechanic load ....................................................... 8
  3.2 The electric circuit ....................................................... 8
  3.3 The control object (MCLite026) .................................... 9
  3.4 The control system interface ....................................... 10
    3.4.1 External interface ................................................ 12
    3.4.2 The input signal path ........................................... 12
    3.4.3 The parameter path ............................................. 14
    3.4.4 The output signal path ....................................... 14
    3.4.5 The FORTRAN ⇒ C interface ............................ 16

4 Simulation results ............................................................ 17

5 Future work ...................................................................... 19

6 References ....................................................................... 20

7 Appendix .......................................................................... 21
  7.1 Operating the drive system ......................................... 21
    7.1.1 Recommended start procedure ............................. 21
    7.1.2 Alternative start procedure ................................. 21
  7.2 Simulation interface modification HOWTO .................. 22
    7.2.1 Add a measured signal. ....................................... 22
    7.2.2 Add a parameter ................................................ 23
1 Introduction

1.1 Goals
This work has two main goals:

1. To develop a skeleton control, which implements most of the basic functionality of a motor drive system, based on a standard HVDC Light™ control. This preliminary base control system should transfer all the specified I/O signals correctly and contain the planned regulators and algorithms. There is no need for control performance optimization.

2. To fit the new control system into the simulation interface for PSCad/EMTDC, to enable further development, testing and optimization.

1.2 Description of application
The inverter control [1] is designed to control a motor drive system, which consists of a voltage-source converter, a synchronous motor (Motorformer™), gearbox and compressor.

Figure 1 shows an overview of the complete drive system, starting with an infinite bus to the left and ending with a compressor to the right. The left (AC→DC) control was used as a starting point for development of the motor drive control.

1.3 Previous work
The HVDC Light™ + Motorformer™ concept was investigated in the spring of 2000 [2]. The study resulted in a HVDC control written entirely in FORTRAN that incorporates the function of a motor drive. The control code has been a valuable reference for this work.

The control presented in this work conforms to the functional description of the motor drive control system [1], which serves as a specification of the control systems functionality. The functional description was presented in the fall of 2001.
2 The converter control system
This section describes the different parts and aspects of the HVDC Light motor drive control system. The control system consists of a number of hardware systems for monitoring, communication and valve control. There is also a number of software systems which defines the functionality of said hardware.

2.1 The control hardware: Mach2
The Mach2 control hardware [3] consists of three subsystems, shown in Figure 2.

- I/O: The I/O units perform sampling, signal conditioning, A/D-conversion and scaling of signals.
- CPU: The main computer consists of two Intel Pentium III processors, system RAM and a 32-bit PCI bus. A PS801 card is mounted on the PCI bus, hosting six SHARC DSP units and an Intel 80486DX processor. The main processors have system supervision functions whereas the regulation, system controls and PWM pattern generation is handled by the DPSs.
- FPGA: This units receives information from the DPSs about when and for how long the switches should be in their conducting state, and emits light impulses to the switch arrays accordingly. FPGA is an acronym of "Field Programmable Gate Array".

Figure 2 Mach 2 control system hardware overview
2.2 The control software: C_system_hidraw

The control software is a generic computer program, written in a hardware-independent symbol-language called HIDRAW. The symbol script code renders source-code for any of a number of standard languages including C for common PC hardware, and low-level assembler for SHARC DSP processors. The procedure of generating source-code for a specific platform is very flexible and offers a wide range of uses for a single version of the software. This opens the possibility for realistic testing of the control system software.

2.2.1 Software functionality

The control system performs a number of tasks designed to control the speed of a motor connected to a voltage source converter. Figure 3 shows an outline of the different tasks and the signals involved. The output signals of the control system are firepulses to the converter bridge, and the field voltage to the magnetizing circuit of the motor. There are three distinguishable subsystems;

1. I/O. This is a set of functions used to read and write both analogue and binary signals to and from peripheral equipment.
2. Signal congestion. This includes scaling, filtering and shaping of raw input, flux estimation, transformations and PWM pattern generation.
3. Feedback loops. The control incorporates a number of regulators:
   - The “Speed and torque control” block contains a regulator for the torque reference
   - The “Flux estimation and control” block contains a regulator for “Im” reference. Im is a fictive current generating the airgap flux.
   - The “Field current control” block regulates the field voltage reference fed to the magnetizing system of the motor.
   - The “Reactor current control” block determines the desired converter voltage according to the current reference. The AC current is controlled by regulation of the voltage drop across the reactor. As the impedances of the motor and the reactor are subject to errors caused by temperature, saturation, flux leakage and linkage, a regulation loop is deployed to obtain high precision of the output current.
2.2.2 Target platforms

The control system software functionality can be realized as a number of different types of executable code, for example:

- A stand-alone binary for uploading to flash RAM on the PS801 card where it can be executed by the DSPs and used in a physical system.

- A "plugin" library for including in some other application such as the simulation binary generated by PSCad, where the core component - EMTDC - calculates states in a set of nodes with electrical components connecting the nodes. One of these components is the HVDC Light™ control system "box", which is a front-end to the control system library. This is the case for normal simulation in PSCad version 3 on the PC platform.

2.2.3 Binary generation

In order to generate a control system library, certain steps must be taken.

1. Develop the control software using HIDRAW. Place symbols on a drawing and connect them with lines. The symbols and lines represents functions (or operators) and variables, respectively. Each symbol in HIDRAW has an included script that generates a piece of source code in C.

2. Generate code for the control with a program called "HidrawWS". This program parses all of the selected drawings and executes the symbol scripts, generating the source code.

3. Make sure all of the source code for the control system is located in the same folder. A simple script named "PC_TO_PC.BAT" copies the files from the different folders to a specific target folder.
4. Load the workspace for the library using a development tool capable of compiling standard C code (usually Visual C/C++ or Visual FORTRAN). The workspace contains information about which source code files to compile and which header files to include. The workspace file is named "C_hidraw.dsw" and is located in the folder "system_software\C_hidraw".

5. Build the library.

The control system code is developed with hardware compatibility and functionality as primary goals. The HIDRAW implementation therefore cannot use other means of communication than those given by the hardware implementation of the control system. This usually means transferring data between the memory of other equipment and the DSP signal namespace. Figure 4 shows the transfer path's of measured signals (left), parameters and operating orders (top) and fire pulse information (right).

![Figure 4 The structure of the physical control system](image)
3 Simulation model

The simulation environment consists of a number of systems that models the behavior of a physical system in a realistic way. This includes all of the hardware systems involved, as well as the software systems that interacts with the control software.

3.1 The mechanic load

The compressor is modeled in steady state as a function \( T(\omega) = -k \cdot \omega^2 \)

where \( T \) is the mechanical torque in pu (always retarding the machine), \( \omega \) is the mechanical angular speed of the machine in pu and \( k \) is an adjustable constant. The transient behavior of the mechanical system is not implemented at this stage.

3.2 The electric circuit

Figure 5 shows the main draft used in this work. It consists of a standard HVDC-Light™ converter with one harmonic filter tuned to the switch frequency, connected via a breaker directly to a synchronous machine without a transformer.

The converter valves (IGBT switches) are located to the left in Figure 5 fed by two ideal voltage sources set at 60 kV. The gate of the switches is connected to the control object, shown in Figure 6 via data labels. The middle part of Figure 5 shows how the reactor and the harmonic filter are connected. The filterbus measurements are also done here. The motor is located to the left in the circuit, connected via a breaker, the current transformers and voltage probes.

The field-winding circuit is located above the motor and the mechanic load is modeled beneath it. It is possible to feed the motor with a constant user-defined field-winding voltage; this also applies to the mechanical torque.
3.3 The control object (MCLite026)

The PSCad draft object definition consists of:

- **Graphics**
  The object graphics with electrical nodes and data nodes is visible from the main draft and is effectively the interface between the main draft and the object code. Figure 6 shows the control system block used in the main draft with its "data labels".

- **Menus, parameters**
  An object may have several parameters controlling different aspects of the object, some are given as signal names (defined in the main draft) instead of values. In the case of a HVDC Light control object, nominal voltage, nominal current and base frequency is typical parameters. These parameters are arranged in a set of menus, which becomes the interface between the operator and the object function.

- **Comments**
  General description of the object, version numbers and miscellaneous comments belongs here.

- **Object code**
  Every object in PSCad has a certain transfer function that defines the output signals as a function of the input signals. This object serves as an interface and does not define any output data itself. It merely acts as a router for I/O data and parameters, which is the task of the object code.

- **A file include list**
  An object can have a list of files to include in the code. The Mclite026 object has several included FORTRAN files, which defines functions that are called from within the object code. These files are referenced with relative paths.

The tool "component workshop" in PSCad is used for altering the definition.
3.4 The control system interface

This section describes some parts of the system libraries C_system and F_system that are relevant to this work, and some parts of the control system draft object.

The control system interface has several parts which performs different tasks. Figure 8 shows four application dependant parts of the interface, all four are connected to the “draft object definition file”. On top of the definition file is a graphic external interface to the main draft and the object parameter menus. Moving downwards, there are three separate data paths; to the left is the “real-time” input signal path, the parameter transfer path is located in the middle and to the right there is a data path for output signals.
Figure 8 Different parts of the simulation interface
### 3.4.1 External interface

Figure 9 shows a part of the graphic external object interface in the component workshop view. At each arrow for input there is a text label which is visible in the main draft, and a signal node with a separate internal label. The node label becomes a valid variable name in the object definition code. The output signals is represented as a line (without the arrow) with a text label and a node label.

![Figure 9 The external interface of the control system object](image)

### 3.4.2 The input signal path

This section describes the input signal path where input data is fed through the simulation interface to the control software. This is shown as the “Object definition file ⇒ MEAS_INTERFACE ⇒ TDM_MEM” path of figure 6.

In order to transfer electrical and mechanical states (signals) from the main draft to the Mach2 control code, a chain of three functions in three files must be called. The object definition file calls a subroutine defined in the file MEAS_INTERFACE.F and passes on voltages, currents and angles. This subroutine calls different functions defined in MEAS_INTERFACE_PACK.C via matching interface entry defined in PSCAD_F2C.F.

The PSCad draft control-block object definition code has a section calling the subroutine MEAS_INTERF( ) with all of the measured signals for further distribution. This section is presented here:

```fortran
! ****** ****** ****** ******************
! CALL MEAS_INTERF($Uac1,$Uac2,$Uac3,$IV1,$IV2,$IV3, &
& $Ud1,$Ud2,$IM_1,$IM_2,$IM_3,$N, &
& $IFF,$THETA,$TIM,DELT)
! ****** ****** ****** ******************
```

The subroutine MEAS_INTERF( ) is defined in the file MEAS_INTERFACE.F, it performs some scaling to simulate the measurement hardware and calls the FORTRAN end of the F2C interface for each set of input data. The subroutine...
header and an example “F2C_F”-function call (for motor-related signals) looks as this:

```fortran
SUBROUTINE MEAS_INTERF(Uac1,Uac2,Uac3,IV1,IV2,IV3,
&   Ud1,Ud2,IM_1,IM_2,IM_3,N,IFF,THETA,TIM,DELT_IN)
! --------------------------------------------------
REAL IMP_1,IMP_2,IMP_3,IM_SCALE
!
IM_SCALE=0.10
IMP_1=IM_1*IM_SCALE
IMP_2=IM_2*IM_SCALE
IMP_3=IM_3*IM_SCALE
!
CALL MEAS_MOTOR_F2C_F(IMP_1,IMP_2,IMP_3,N,IFF,THETA)
!
! ---------------------------------------------------
```

The name of the function to be called is always ended with "_F2C_F" to clearly state that this is the FORTRAN part of the FORTRAN→C interface that is called. The other "side" of the interface is ended with "_f2c" and is defined in the C_system library.

Presented below is the section of the file MEAS_INTERFACE.C, defining the function `meas_motor_f2c()` and writing the signals to the memory of the I/O device, i.e. the control code:

```c
/* ********* Input signals to DSP 1 **************/
meas_motor_f2c(double infc1, double infc2, double infc3,
   double infc4, double infc5, double infc6)
{

   /* *****************************************/
   //
   /* *****************************************/
   // N to DSP1
   TDM_MEM[22] = infc1;
   TDM_MEM[23] = infc2;
   TDM_MEM[24] = infc3;
   TDM_MEM[25] = infc4;
   TDM_MEM[26] = infc5;
   TDM_MEM[27] = infc6;

   //
}

} // end of meas_motor_f2c
```

When the signal has entered the TDM_MEM vector, it is readable by a HIDRAW symbol called "Auto serial port". The symbol has two output channels and is configured to read from a certain slot. The slot number is a parameter inside the symbol, the resulting code is basically

```c
Output_0_name = TDM_MEM[slot_number*2 + 0];
Output_1_name = TDM_MEM[slot_number*2 + 1];
```
So in the example case in `meas_motor_f2c()`, TDM_MEM[22] corresponds to slot 11 and output 0 of the symbol.

### 3.4.3 The parameter path

This section describes the “Object definition code ⇒ PCI_PACK.C ⇒ HOSTIN ⇒ DSP” path of Figure 8.

The object definition code puts the menu parameters into the STOR( ) vector at specific positions. These positions are allocated manually by the developer.

PCI_PACK.C defines the functions `pci_stor2mem()` and `pci_mem2stor()` that copies parameters between the Emtdc STOR( ) runtime data vector and the namespace memory of the hostin application.

The HOSTIN application then copies the parameters from its own memory space to the appropriate DSP at a certain rate controlled by the main XEX timer (adjustable through `XEX_TIMER.F` in `System_software\F_system_source`).

A parameter stored in the STOR( ) vector is only a value and has no name. It is crucial that the functions in PCI_PACK.C respects the parameter allocation made in the object definition code. A position mismatch between the object code and the two functions in PCI_PACK.C results in uncontrolled data corruption.

### 3.4.4 The output signal path

This section describes in how the output signals (fire pulses and field voltage signal) gets transported from the control code out to the simulation draft. This is the “Object definition file ⇒ VSC_CPG ⇒ TDM_PACK ⇒ FPGA_DPM” path of Figure 8.

Each simulation timestep, a section of the PSCad object definition code calls the subroutine `VSC_CPG()`. This section is shown below:

```!
CALL VSC_CPG($FP,$FT,$UFF,CHOP,DspInt,$DEBL)
!
```

The subroutine `VSC_CPG( )` is defined in the file `VSC_CPG_CHOP.F`. Below is an outline of the subroutine header and some of the `FPGA_C2F_F( )` calls:

```!
*--------------------------------------------------
SUBROUTINE VSC_CPG(FPY,FTY,UFF,CHOP,DspInt,DEBLOCK)
DO INDEX=1,6
   FPY(INDEX) = NINT(FPGA_C2F_F(INDEX))
   FTY(INDEX) = FPGA_C2F_F(INDEX+13)
ENDDO
!
!
   DSP_INT = FPGA_C2F_F(13)
   UFF=FPGA_C2F_F(20)
!
*--------------------------------------------------
```
The subroutine FPGA_C2F_F() is called several times. On the other side of the F2C interface there is a function written in C that maps the index to a set of data located in the FPGA application, which holds information about the fire pulses generated by the control system. It also returns an analogue output signal (UFF, the field-winding voltage), which is defined by DSP1 and written to the STOR() vector directly.

This is how the function fpga_c2f() is defined in the file TDM_PACK.C:

```c
double fpga_c2f(int index)
{
/*********************************************/
// Transfer FP-pulses from FPGA to EMTDC
k_pnt = index;
if (k_pnt <= 20 ) {

    switch (k_pnt-1) {
        case 0: out_value = FPGA_FP1;break;
        case 1: out_value = FPGA_FP2;break;
        case 2: out_value = FPGA_FP3;break;
        case 3: out_value = FPGA_FP4;break;
        case 4: out_value = FPGA_FP5;break;
        case 5: out_value = FPGA_FP6;break;
        case 6: out_value = FPGA_DPM[16];break;
        case 7: out_value = FPGA_DPM[17];break;
        case 8: out_value = FPGA_DPM[18];break;
        case 9: out_value = FPGA_DPM[19];break;
        case 10: out_value = FPGA_DPM[20];break;
        case 11: out_value = FPGA_DPM[21];break;
    }
}
return(out_value);
}
```

This code abstract in this section shows how the function call is used to transfer different portions of the FPGA memory namespace to the object code namespace depending on the index value. This address mapping is defined at two separate locations; both in the file VSC_CPG_CHOP.F - subroutine VSC_CPG(), and in the file TDM_PACK.C – function fpga_c2f().
3.4.5 The FORTRAN ⇒ C interface

This section describes how the interface file PSCAD_F2C.F is constructed.

An entry in PSCAD_F2C.F generally looks like this:

! called from MEAS_INTERF
SUBROUTINE MEAS_MOTOR_F2C_F(R1,R2,R3,R4,R5,R6)
REAL R1,R2,R3,R4,R5,R6
INTERFACE
SUBROUTINE MEAS_MOTOR_F2C(R7,R8,R9,R10,R11,R12)
REAL R7,R8,R9,R10,R11,R12
!DEC$ ATTRIBUTES C :: MEAS_MOTOR_F2C
END SUBROUTINE
END INTERFACE
CALL MEAS_MOTOR_F2C(R1,R2,R3,R4,R5,R6)
END SUBROUTINE

A subroutine written in FORTRAN code cannot directly call a function written in
C due to different semantics and data types, therefore an interface file -
PSCAD_F2C.F - translates the calls.

The translation mainly consists of a new function carrying a compiler directive
"!DEC$ ATTRIBUTES C", which tells the compiler that this subroutine has the
same properties as a function written in C would have. This new "C" function then
gets called with the values of the first (R1, R2...) set of variables. [4]
4 Simulation results

The control code and its simulation interface is fully functional and is capable of running as intended. Figure 10 shows a typical simulation situation. At the top of Figure 10 is the control panel for some of the simulation objects, containing a number of switches. From left to right:

- Two controls for the synchronous machine. The first switch controls the behavior of the rotor (running free or locked to nominal speed), the second switch enables the SM model to act as a machine instead of an infinite voltage source.

- Three controls for the load simulation. The left slider sets a constant mechanic torque, the right slider sets the multiplier constant for the speed-dependant torque calculation and the switch in the middle controls which of the torque values is to be used.

- The motor breaker switch. This switch controls whether the main circuit breaker is closed or open.

- Three controls for the speed control. The left slider sets the constant electric torque reference in the motor drive control, the right sets the speed reference in for the speed regulator. The speed order value is given as the voltage frequency on the bus. The switch in the middle controls if the speed regulator is operational. If not, the control system runs in constant torque mode.

- Two controls for the field voltage regulation. The switch controls whether the field voltage signal from the control system is fed to the motor or if the constant value from the slider is used instead.

- The valve control deblock switch.

The graphs of Figure 10, top-down:

- Active and reactive power fed into the motor, in [MW].
- Motor speed, in [pu].
- Mechanical and electric torque. The values are presented in [pu], with different references; a positive mechanical torque accelerates the shaft, a positive electric torque decelerates the shaft.
- The bus voltages, in [kV].
- The bus currents, in [kA].
- The field-winding current, in [pu].

The x-axis common for all the graphs, it is graded in seconds from the start of the simulation. At time=5 seconds, the frequency order is changed from 85 hz to 80 hz. The speed-order step causes all of the regulators to act, which is visible in the runtime graphs.
Figure 10 Run-time simulation of a speed-order step from 85 to 80 hz.
5 Future work
The following parts of the project need further development.

- The DSP partitioning. Currently all of the control system code is placed in DSP1, which works fine in the simulated environment but breaks hardware-compatibility; each DSP has a limited number of analog input channels and a limited amount of processing power. The control code needs to be split up and the parts employed in different DSPs.

- HIDRAW page naming. Some of the HIDRAW pages have altered to perform completely different tasks but their names remain unchanged and are therefore wrong. The new pages have temporary, possibly misleading names. New explanatory names should be applied to these pages as soon as possible in order to avoid confusion.

- There will be problems running both the standard rectifier control and the modified converter control in the same simulation draft. These problems has not yet been identified but are related to name separation of different versions of library functions. A temporary but viable solution is to add up the interfaces into one general interface capable of running both sides, and include the DSP code from both stations into one DSP application.
6 References

References 1-3 is subject to non-disclosure paragraphs.

[1] ABB internal document
[2] ABB internal document
[3] ABB internal document
   Katarina Blom, Studentlitteratur 1994
   ISBN 91-44-47881-X
7 Appendix

7.1 Operating the drive system

7.1.1 Recommended start procedure

Make sure that:

1. The motor breaker is in the “open” state.
2. The rotor is locked.
3. The control is in constant torque mode (0.0 pu)

Start the simulation. As the control builds up the motor flux, the pole voltage increases. During this period the control tries to feed a reactive current into the machine in order to supply stator flux, which results in a relatively high voltage. Since the SM has difficulties handling this current surge, the breaker must remain open until the flux is close to its nominal value. If the machine becomes unstable when the breaker is closed, open it again immediately and wait for the machine to settle.

When the breaker has been successfully closed, the rotor may be unlocked and the control torque adjusted to maintain the speed of the machine. The regulation in version 1.0 of the control seems to have problems when the control system tries to increase the speed of the motor.

When this balanced steady state is reached, it should be possible to send the control into constant speed mode. In case of large currents and/or unstable operation upon switching to speed regulation mode, set the target speed to a value lower than the actual speed of the machine.

7.1.2 Alternative start procedure

This way of starting the simulation is slightly faster. Also the motor breaker is closed throughout the simulation, keeping the regulation loops intact.

1. Close the motor breaker.
2. Unlock the rotor
3. Set the control to constant torque mode (0.0 pu)

Start the simulation. This procedure allows reactive current to flow into the motor, assisting the flux build-up. Since the motor model does not have the capability to catch fire or shorten its usable life span, this method can be found useful. An advantage of this method is the fact that the breaker is closed throughout the simulation, keeping the regulation feedback loops intact.
7.2 Simulation interface modification HOWTO

7.2.1 Add a measured signal.

The following steps needs to be considered in order to create a new path for real-time measured analogue signals through the interface.

1. Edit the graphics of the object symbol in "component workshop". Copy an existing arrow and data label, and paste it at a free space somewhere around the object. Name the node label "EXAMPLE _1".

2. Edit the object definition code. Add "$EXAMPLE_1" to the parameter list in the call to MEAS_INTERF(). Note the new order of the parameters.

3. Edit MEAS_INTERF.F in the "System_software\F_system_source" folder. Update the header defining the subroutine MEAS_INTERF() according to the new set of parameters. The order of the parameters is essential and must be equal to the call in the object definition code.

4. Create a new subroutine call in MEAS_INTERFACE.F like this:

   !
   call MEAS_EXAMPLE_F2C_F(EXAMPLE_1)
   !

5. Edit PSCAD_F2C.F and add a new entry for MEAS_EXAMPLE_F2C_F. Copy an existing entry and change the names of the subroutines. The headers needs to accommodate all the parameter the subroutine gets called with.

6. Edit MEAS_INTERFACE.C in the "System_software\C_system_source" folder, adding the function "meas_example_f2c()" in which some data vector (reachable by some symbol in the HIDRAW code) is assigned the signal value.

7. Add the signal to the HIDRAW code using some suitable symbol and a connecting line.

If necessary, add more signals to the newly created F2C interface entry and the functions and function calls. Each function can accommodate a large number of signals and the naming is used mainly for programming clarity issues.
7.2.2 Add a parameter.

These steps need to be taken in order to add an example parameter to the simulation interface:

1. Add a parameter entry box in the parameter section of "component workshop". Name the parameter "EXAMPLE_2".

2. Add a parameter assignment entry in the list of "STOR(NPOS + XX) = $PARAMETER_NAME", put the entry at the end of the list for convenience. Add one to the XX number the new entry.

3. Edit PCI_PACK.C in the "System_software\C_system_source" folder. Add the new parameter in the parameter declaration list, in the functions pci_mem2stor( ) and pci_stor2mem( ). Be observant of the addressing in the copy procedure to/from the STOR vector!

4. Rebuild C_system.lib

5. Edit the HOSTIN application defined in "System_define\Hostin". Add a declaration of the new parameter on one of the init pages, using a numeric constant symbol with a line attached. Name the line "IN_EXAMPLE2". Then proceed to one of the settings pages and add an external reference to the newly created signal, a simple repeater symbol and a "DSP Send" symbol. The line connecting the repeater and DSP_SEND should be called "EXAMPLE2".

6. Edit the receiving DSP application. Create the new signal named "EXAMPLE2", using a numeric constant symbol in one of the settings pages.

7. Rebuild the HIDRAW code and run PC_2_PC.BAT in order to gather all the source code to C_hidraw_source.