Integrating Third Party Tools Using the C Interface

March, 2002

PARADIGM™ WORKS

Overview

- What is the C interface?
- Why would you need such an interface?
- How do you use the interface?
- Conclusion
- Questions and maybe some answers

PARADIGM™ WORKS

What is the C interface

- A collection of C macros that enables C functions to be called from within Specman Elite
  - SN_TYPE
  - SN_ENUM
  - SN_LIST_NEW
  - Many others

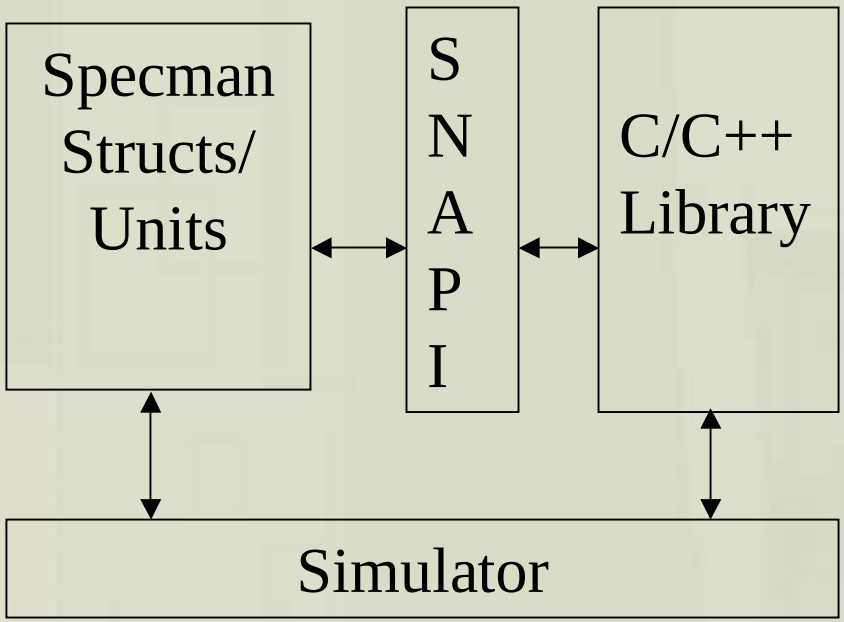
PARADIGM™ WORKS

Why use it?

- Legacy code
- Third party packages
- Performance modeling
- Custom code

PARADIGM™ WORKS

Component View



PARADIGM™ WORKS

The Process

- C++ methods and/or C functions.
  - The library API
  - The wrapper method/function
- Maping Specman method calls.
- The Makefile and linking (a.k.a the hard part)
  - Static vs. Dynamic (example uses dynamic)
  - Targets – compiling and linking in stages
- Run the simulator

PARADIGM™ WORKS

Simple Example

- A C++ method that returns a reference to an instance of a class.
- The *e* struct that utilizes this C++ method.

PARADIGM™ WORKS

The C++ code

```
#include "api_class.h" // GenApiClass and Result_t
#include "sn_interface.h" // SN_TYPE

extern "C"
{
  /* This function instantiates a generator API class with
     a name passed via the argument list*/

  unsigned int tb_gen_init( SN_TYPE(string) instance_name ){
    GenApiClass *GenApi; // declare a pointer to the object
    Result_t     Result; // and a return type
    GenApi = new ApiClass(instance_name,&Result);
    if (Result == R_PASS) {
      printf("Interface suceeded in linking GenComp\n");
      printf("GenComp is located at %p\n",GenApi);
    }  else  {
      printf("Interface FAILED in linking GenComp\n");
      exit(2);
    }
    return((unsigned int)GenApi);
  }
};
```

PARADIGM™ WORKS

The *e* code

```
<'
// define method to function mappings. Routines must be compiled
routine tb_gen_init(instance_name: string):uint is C routine tb_gen_init;

struct MyObject {
   !name : string;
   !p_object : uint;
};

unit tb_u {
   object : MyObject;

   run()is also {
      object.name = "GenComp";
      object.p_object = tb_gen_init(g.name);
      out("MyObject ",object.name," resides in memory at ",object.p_object);
   };
};

extend sys {
    tb : tb_u is instance;
};
'>
```

PARADIGM™ WORKS

Another Example

- A C++ method that uses a Specman Struct.
- Specman supports a limited number of parameters in the routine call.

PARADIGM™ WORKS

C++ Code

```
#include "api_class.h" // GenApiClass and Result_t
#include "sn_interface.h" // SN_TYPE

extern "C"
{
  void tb_gen_set_source(SN_TYPE(S_set_source) Source, unsigned int addr ){
    GenApiClass *GenApi; // declare a pointer to the object
    Result_t     Result; // and a return type
    GenApi = addr;
    printf("Switching source to %s with pattern %s\n",
                Source->source, Source->Pattern);
    Result = GenApi->Set_Source( Source->Spen,
                                 Source->source,
                                 (TSrcMode) Source->Mode,
                                 /*
                                  ...
                                 */
                                 (TPIBoolean)Source->B1,// cast as TPI type
                                 (TPIBoolean)Source->B2 // cast as TPI type
                                 );
    if (Result == R_PASS) {
      printf("Succeeded in calling Set_Source\n");
    } else {
      printf("Set_Source failed with %i\\n",Result);
      exit(2);
    }
  }
};
```

PARADIGM™ WORKS

e Code

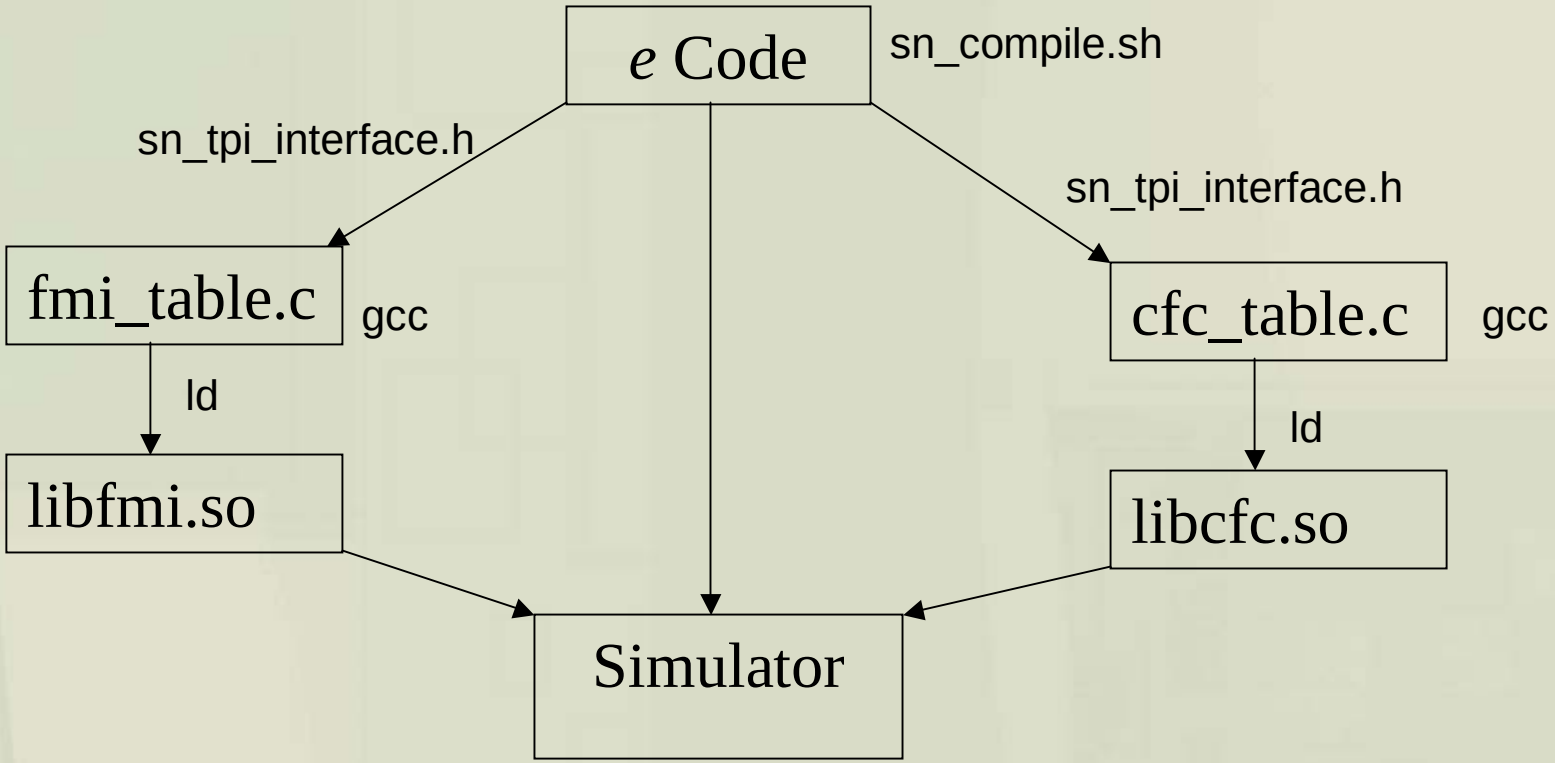
```
<'
// define method to function mapping
routine tb_gen_set_source(source: S_set_source, addr: uint) is C routine tb_gen_set_source;

struct S_set_source {
   !Spen: int;
   !source:  string;
   !Mode: SrcMode;

   // …

   !B1: bool;
   !B2: bool;
};
'>
```

Calling the Method

```
<'
extend tb_u { // extend the tb from the previous example
  event clk is rise('~/clock')@sim;

  run() is also {
     start tcm();
  };

  tcm()@clk is {
     var source : S_set_source;
     // set the struct to the new values
     source = new S_set_source with {
                                        .Spen = 1;
                                        .source = "My_String";
                                        // ...
                                        .B1 = FALSE;
                                        .B2 = FALSE;
                                        };

     tb_gen_set_source(source, object.p_object);
     wait [1]*cycle;

     stop_run();
  };
};
'>
```

PARADIGM™ WORKS

Linking the Simulator



PARADIGM™ WORKS

Compiling *e*

e Code

- sn_tpi_interface.h
- sn_tpi_interface_pic_.o
- sn_tpi_interface__main_pic_.o

```
# Create header file which used by C-interface
sn_twb_interface.h: sn_twb_intermediate
     cp t_intermediate/sn_twb_interface_.h $@

# Create Specman header file and binary objects from e-source
sn_tpi_intermediate:
     if [ ! -d t_intermediate ]; then mkdir t_intermediate; fi
     sn_compile.sh -t t_intermediate ../e_src/sn_tpi_interface.e -pic -no_link
```

PARADIGM™ WORKS

Foreign Model Interface Library

fmi_table.c

libfmi.so

- This is a dynamically linked library used by ncvhdl

```
 # Create binary object from C-interface source file
 sn_tpi_interface.o: sn_tpi_interface.h
       gcc -traditional-cpp -c -fPIC -D_NO_PROTO -o sn_tpi_interface.o \
       -I${CDS_INST_DIR}/tools/inca/include -I${TPI_INSTALL_DIR}/src/include \
       -I${TWB_INSTALL_DIR}/src/include/extern -I . ${C_DIR}/sn_tpi_interface.cpp

# Create binary from Specman/3rd-party foreign model interface(FMI) table
fmi_table.o: ${C_DIR}/fmi_table.c
      gcc -c -fPIC -D_NO_PROTO ${C_DIR}/fmi_table.c -I${CDS_INST_DIR}/tools/inca/include


# Create Specman/3rd-party FMI library which dynamically loaded by ncsim
libfmi.so: fmi_table.o sn_tpi_interface.o
      ld -G -z text -o $@ fmi_table.o sn_tpi_interface.o \
         t_intermediate/sn_tpi_interface_pic_.o \
         t_intermediate/sn_tpi_interface__main_pic_.o \
         ${SPECMAN_HOME}/solaris/ncvhdl_sn_pic_.o \
         ${TPI_INSTALL_DIR}/sun4Solaris/lib-gccsparcOS5/libfmi.so -lsn
```

PARADIGM™ WORKS

C Function Call Library

- This is a dynamically linked library used by ncvhdl

```
# Create binary from Specman C function call(CFC) library table
cfc_table.o: ${C_DIR}/cfc_table.c
      gcc -c -fPIC -D_NO_PROTO ${C_DIR}/cfc_table.c -ICDS_INST_DIR}/tools/inca/include



# Link CFC library which is dynamically loaded by ncsim
libcfc.so: cfc_table.o sn_tpi_intermediate
      ld -G -z text -o $@ cfc_table.o \
         t_intermediate/sn_tpi_interface_pic_.o \
         t_intermediate/sn_tpi_interface__main_pic_.o \
         ${SPECMAN_HOME}/solaris/ncvhdl_sn_pic_.o -L${SPECMAN_HOME}/solaris -lsn
```

PARADIGM™ WORKS

Simulation

Simulator

```
# Create Specman stub, compile and elaborate all VHDL files
vhd_compile:
      rm -rf ../work
      mkdir ../work
      specman -c "write stubs -ncvhdl"
      ncvhdl -mes -v93  -cdslib ./cds.lib -work work -componly e  ./specman_nc.vhd -append
      ncvhdl -mes -v93  -cdslib ./cds.lib -work work -componly a  ./specman_nc.vhd -append
      ncvhdl -mes -v93  -cdslib ./cds.lib -work work -componly e  ../tb/tb.vhd -append
      ncvhdl -mes -v93  -cdslib ./cds.lib -work work -componly a  ../tb/tb.vhd -append


sim:    libfmi.so libcfc.so vhd_compile
      ncsim -gui -cdslib ./cds.lib -logfile ncsim.log  -errormax 15  -messages \
            -status work.DEMO1:DEMO1  -STACKSIZE 1000000


# Remove all intermediate files
clean:
      rm -rf sn_twb_interface.o *.so sn_twb_interface.h *.c t_intermediate
```

PARADIGM™ WORKS

Review

- Create Specman header file with sn_compile.sh
- Compile custom source code
- Link shared objects with simulator
- Create our specman stub file
- Compile our hdl code
- Run the simulator with dynamically linked libraries

PARADIGM™ WORKS

Conclusion

- The C interface lets you leverage code you already own.
- The hard part is getting all of the objects to link.
- DON'T store references to *e* objects in C/C++
  - Garbage collection will cause null pointers and lots of headaches
- Where to get help
  - Chapter 10 of Usage and Concept Guide
  - Chapter 14 of Usage and Concept Guide
    - Specifically 14.8.2 – 14.8.6
  - Apendix C of Usage and Concept Guide
  - Your local Verisity CE

PARADIGM™ WORKS

Questions

- Contact information:

  John.Morris@Paradigm-Works.com

- For more info on Paradigm Works
  - Consulting Services and Technology

    http://www.Paradigm-Works.com