Gnucap

a user extendable simulator

Outline

- History
- Algorithms
- Issues
- Plugins ---------->
- Work in progress
- Future work

- Plugin concepts
- Plugin types
- Device models
- Spice wrapper

History

- Started late 70's, home computer, CP/M
- Design exploration, audio, RF
- PhD – implicit mixed mode simulation
- Modelgen (early model compiler)
- Research in a non-research environment

Algorithms

- Basis is Spice-like
- Mixed-signal
  - Smart nodes, automatic connect modules (1990)
- Fast-Spice
  - Queue driven analog
  - Fast incremental solver

Issues for this talk

- Environment for experimentation
- User contributions
- User customization, while still using RPM or DEB
- Support for new models, without rebuilding
  - High level, porting Spice models
- Support for different simulation languages
  - Spice, Spectre, Verilog
- Extensions in general

Plugins

- Basic concepts
- Plugin types
- Device models
- Wrappers for models written for others

Plugins – basic concepts

- A plugin is a standard shared-object (.so, .dll)
- Based on C++ derived classes
- The dispatcher.
-
- Core has a library functions, database
- Everything else is plugins
- Can be loaded/unloaded by user, interactively

Plugins are shared object modules

- Posix standard “dlopen” interface.
- For non-posix, map the command.
- User command to load and unload.

C++ derived classes

- Core provides a base class, determines type.
- Make a new class, derived from it.
- Plugin namespace is private.
- No symbols are exported.

C++ derived classes

base.h:

```
class BASE {
public:
  virtual void do_it();
};
```

plugin.cc:

```
#include "base.h"

class PLUG : public BASE {
public:
  void do_it() {
    std::cout << "do something here!!\n";
  }
};
```

The dispatcher

- New subclasses are registered with dispatcher
- by name
- C++ map
- One static object is constructed on loading
- Dispatcher has pointer to it
- Look up by name, access through pointer
- Clone if needed.

The dispatcher

plugin.cc:

```
#include "base.h"

class PLUG : public BASE {
public:
  void do_it() {
    std::cout << "do something here!!\n";
  }
} an_instance;

DISPATCHER<BASE>::INSTALL
d(&dispatcher, “name”, &an_instance);
```

Plugin types

- Devices
- Commands
- Parameter functions
- Measure functions
- Behavioral modeling functions
- Languages
- Outputs

Wrappers

- Spice-wrapper (working)
  - can use Spice C models with no modifications
- System-C wrapper (just starting)

- Wraps code written for something else
- Maps the interface

Spice-wrapper – wrapper.h

```
extern "C" {
  #include "bsim4def.h"
  #define DEV_bsim4
  #include "bsim4itf.h"
}
#define info  B4info
#define INSTANCE BSIM4instance
#define MODEL     BSIM4model
#define SPICE_LETTER "M"
#define DEVICE_TYPE "bsim450|bsim4"
#define MIN_NET_NODES 4
#define MAX_NET_NODES 4
#define INTERNAL_NODES 8
#define MODEL_TYPE "nmos14|pmos14|nmos54|pmos54"
static std::string port_names[] = {"d", "g", "s", "b"};
static std::string state_names[] = {"vbd",   "vbs"};
```

Spice-wrapper -- Makefile

```
TARGET = bsim450
$(TARGET):

include makedefs

HDRS = bsim4def.h  bsim4ext.h  bsim4itf.h wrapper.h

SPICE_VERSION = SPICE_3f

include ../Make2
```

Work in progress, future work

- Icarus verilog
  - Step 1: simple call to the Icarus virtual machine
  - Step 2: use as model compiler
- System C as an extension language
  - It's already C++, map the interface
- ADMS
  - Step 1: use it with Spice-wrapper (ugh)
  - Step 2: generate native code