DESIGN & VERIFICATION DVCON 2014 CONFERENCE & EXHIBITION

SystemVerilog, Batteries Included

A Programmer's Utility Library for SystemVerilog

Jonathan Bromley, André Winkelmann

verilab

Why is it *so* hard in SV...

- to read an environment variable?
- to find what files exist in a directory?
- to find the current date and time?
- to do regular expression matching and substitution?
- to read and write configuration files?

Let's make it easier! - *1*

- to read an environment variable
- to find what files exist in a directory

```
import svlib_pkg::*;

string cfgDir = "../cfg";  // set up a default
string cfgVar = "SIM_CFG_DIR";

if (sys_hasEnv(cfgVar))
  cfgDir = sys_getEnv(cfgVar);
...
Pathname path = Pathname::create(cfgDir);
path.append("*.cfg");
...
string cfgFiles[$] = file_glob(path.get());
```

../cfg/*.cfg

Let's make it easier ! - 2

- Wall-clock time, and timestamps of existing files:

```
... cfgFiles = file_glob(path.get());

longint chosenFileTime = sys_dayTime() - 24*60*60;
string  chosenFile = "";
foreach (cfgFiles[i]) begin
  longint fileTime = file_mTime(cfgFiles[i]);
  if (fileTime > chosenFileTime) begin
    chosenFileTime = fileTime;
    chosenFile = cfgFiles[i];
  end
end
if (chosenFile != "")
  $display(sys_formatTime(chosenFileTime, "got %c"));
```

one day ago

```
got Tue Mar  4 16:04:34 2014
```

Let's make it easier ! - *3*

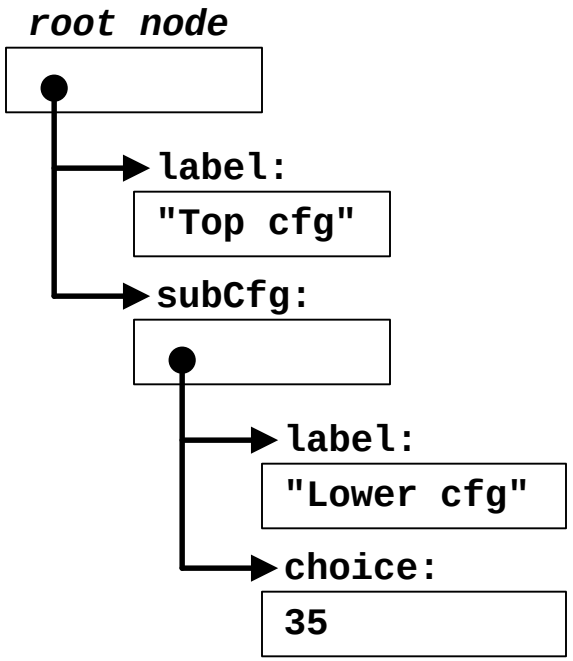
- Read a configuration file in **`.ini`** or YAML format:
  - **Step 1**:
    Read file into format-agnostic Document Object Model

```
label="Top cfg"
[subCfg]
label="Lower cfg"
choice=35
```

chosenFile

```
cfgFileINI ini;
cfgNode    cfgDOM;

ini = cfgFileINI::create();
cfgDOM = ini.readToDOM(chosenFile);
```



Let's make it easier ! - *4*

- Read a configuration file in **`.ini`** or YAML format:
  - **Step 2**:
    Populate user objects from DOM
  - manually or automagically

```
label="Top cfg"
[subCfg]
label="Lower cfg"
choice=35
```

chosenFile

```
GlobalCfg topConfig = new;
topConfig.fromDOM(cfgDOM);
```

beware magic!

```
class GlobalCfg...;
  LocalCfg subCfg;
  string   label;
  ...
```

"Top cfg"

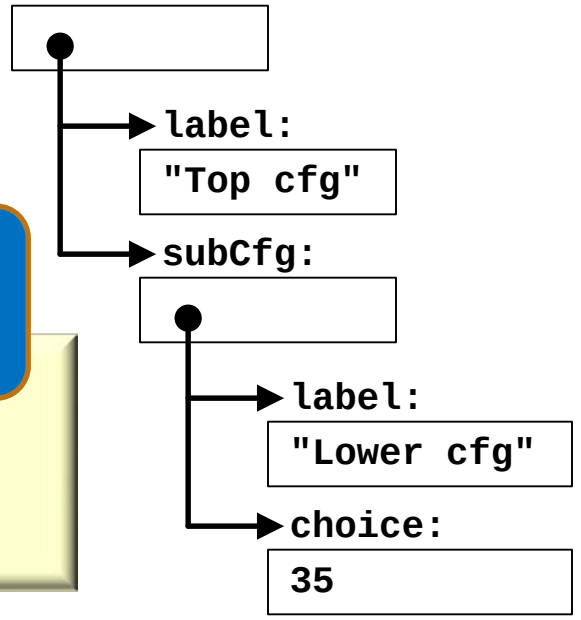
```
class LocalCfg...;
  int    choice;
  string label;
  ...
```

35

"Lower cfg"

user's config classes

root node



Other features - *1*

- Regular expressions, submatches and substitution:

```
Regex re;
re = regex_match( "03/04/14", "([0-9]+)/([0-9]+)/([0-9]+)" );
if (re != null) begin
  $display("Looks like a date");
  void'( re.subst("$2-$1-20$3") );
  $display("UK-style date = %s", re.getStrContents());
```

```
Looks like a date
UK-style date = 04-03-2014
```

- Global substitution:

```
  re.setRE("-");
  if (re.substAll(" ++ ") == 2)
    $display(re.getStrContents());
end
```

```
04 ++ 03 ++ 2014
```

Other features - 2

- File pathname manipulation utilities

```
Pathname path = Pathname::create("/home/svlib/src/svlib_pkg.sv");

$display(path.extension());
```
```
.sv
```
```
$display(path.dirname());
```
```
/home/svlib/src
```
```
$display(path.tail());
```
```
svlib_pkg.sv
```

- Full *stat* information on any file:

```
sys_fileMode_s fm = file_mode("some/file/somewhere");
if (fm.fType & fTypeDir) $display("it's a directory");
```

Key design decisions

- Choice of initial feature set
- Objects or simple functions?
- Choice of error handling mechanism
- Never disturb random stability

- Guiding principle:

natural and expressive for SV programmers

SV can be unhelpful…

- Some SV types offer method-like operations…

```
string S = "  Some text  ";
int n = S.len();
S = S.toupper();
```

these look like methods on object S

- … but native SV types cannot be extended!

```
S = S.trim();
```

svlib Str class method

```
Str ss = Str::create("  Some text  ");
ss.trim(Str::RIGHT);
$display( "\"%s\", ss.get() );
```

`"  Some text"`

svlib package-level function

```
$display( "\"%s\", str_trim(S, Str::BOTH) );
```

`"Some text"`

Objects or plain functions?

- Some utilities work best as simple functions:

```
string nameList[$] = {"DVCon", "Andre", "Jonathan"};
$display(str_sjoin(nameList, ":-:"));
```

```
DVCon:-:Andre:-:Jonathan
```

- For bigger tasks it's better to have an object:

```
Regex re  = Regex::create("an(.)");
int   pos = 0;
re.setStrContents("and another banana");
while (re.retest(.startPos(pos))) begin
  $display("'an' followed by '%s' at pos=%0d",
        re.getMatchString(1), re.getMatchStart(1));
  pos = re.getMatchStart(0) + re.getMatchLength(0);
end
```

```
'an' followed by 'd' at pos=2
'an' followed by 'o' at pos=6
'an' followed by 'a' at pos=15
```

In many cases
svLib offers both options

Error handling

```
longint t = file_mTime("MISSING/FILE");
```

```
<Assertion>: Failed to stat "MISSING/FILE", errno=2
    (No such file or directory)
```

- Optionally, manage errors in user code:

```
error_userHandling(1);
t = file_mTime("MISSING/FILE");
if (error_getLast() != 0)
  $display("whoops, my bad: %s", error_fullMessage());
```

No message, returns t=0

- Error recording and handling **per SV process**
  - Localized effect
  - Fully deterministic

Random stability

- svlib generates objects – may impact random stability
  - Especially troublesome during debug

```
Regex re  = Regex::create("an(.)");
```

never call **new( )** directly

- Managed object creation to preserve random stability
  - Maintains pool of reusable objects for efficiency
  - Ready for future addition of an object factory

```
`ifdef SVLIB_NO_RANDSTABLE_NEW
result = new();
`else
std::process p = std::process::self();
string randstate = p.get_randstate();
result = new();
p.set_randstate(randstate);
`endif
```

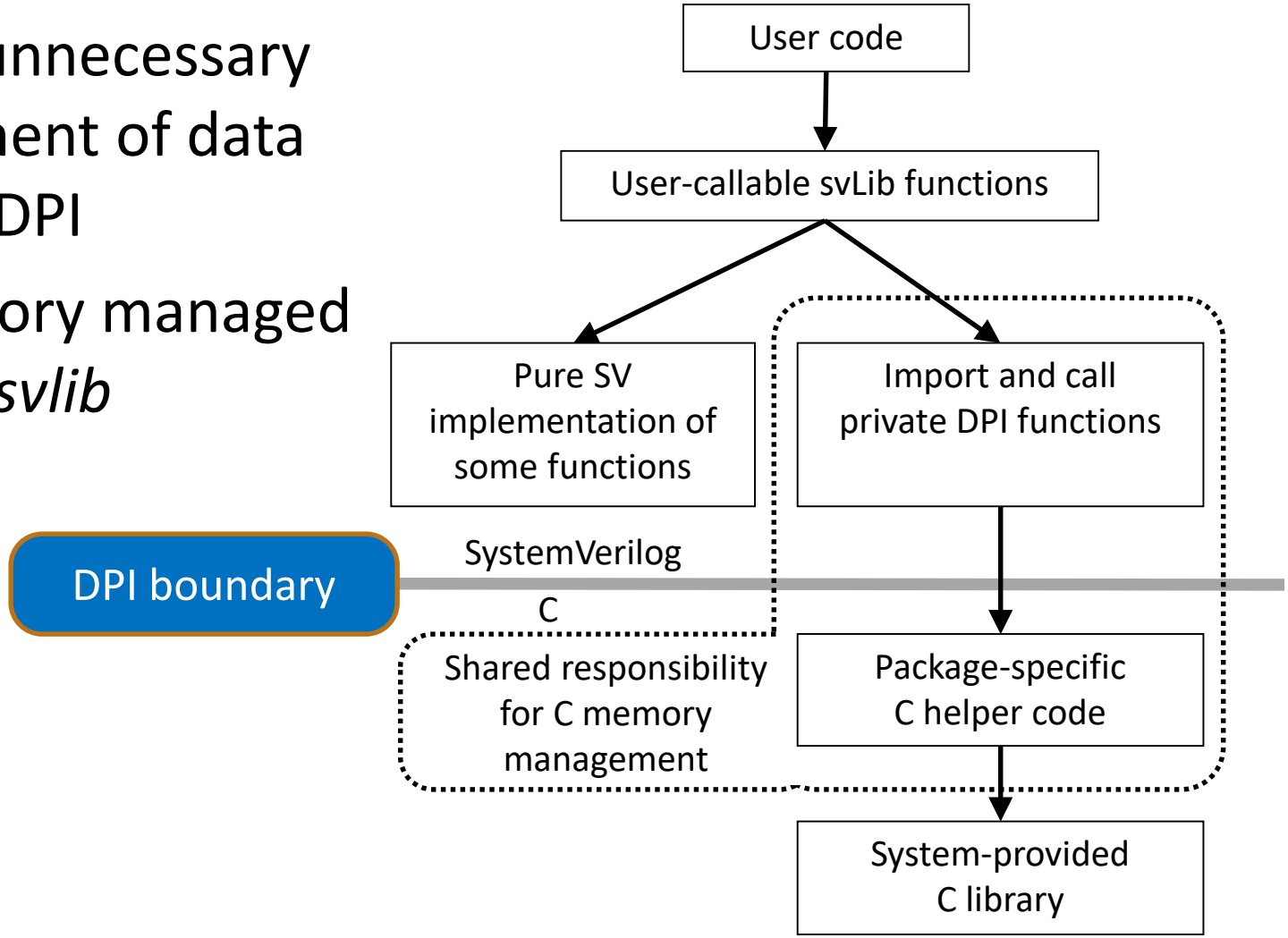
Implementation objectives

Full details in the published paper

- Use external libraries as far as possible
  - robust, well-proven solutions
- Test early, often, and on many platforms
- Managed DPI interface, no memory leaks
  - No DPI functions directly exposed to users
  - DPI memory managed internally by svLib
- Never throw an error from C code
  - All errors reported back to SV and handled there
- Design for performance
  - Avoid unnecessary movement of data across DPI

Implementation architecture

- Avoid unnecessary movement of data across DPI
- C memory managed within *svlib*



Reminder: DOM↔object

- Read a configuration file in `.ini` or YAML format:
  - **Step 2**:
    Populate user objects from DOM
  - manually or automagically

```
label="Top cfg"
[subCfg]
label="Lower cfg"
choice=35
```

chosenFile

```
GlobalCfg topConfig = new;
topConfig.fromDOM(cfgDOM);
```

beware magic!

```
class GlobalCfg...;
  LocalCfg subCfg;
  string   label;
  ...
```

"Top cfg"

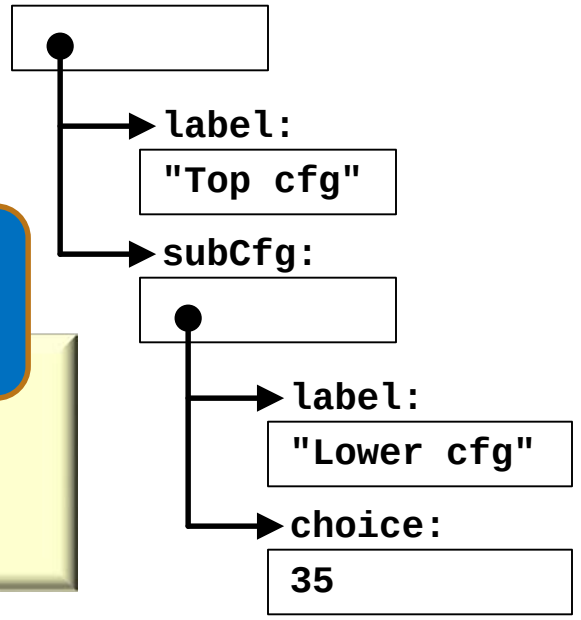
```
class LocalCfg...;
  int    choice;
  string label;
  ...
```

35

"Lower cfg"

user's config classes

root node



Beware – Macro Magic!

- Read a configuration file in **`.ini`** or YAML format:

```
cfgFileINI ini = cfgFileINI::create();
cfgNode cfgDOM = ini.readToDOM(chosenFile);

GlobalCfg cfg  = GlobalCfg::type_id::create();
cfg.fromDOM(cfgDOM);
```

```
class LocalCfg extends ...;
  int    choice;
  string
  `SVLIB
   `SVLI
   `SVLI
  `SVLIB
endclass
```

```
class GlobalCfg extends ...;
  LocalCfg subCfg;
  string   label;
  `SVLIB_DOM_UTILS_BEGIN(GlobalCfg)
   `SVLIB_DOM_FIELD_OBJECT(subCfg)
   `SVLIB_DOM_FIELD_STRING(label)
  `SVLIB_DOM_UTILS_END
endclass
```

Use SV-2012 interface classes?

- Config class can *implement* an interface class:

```
class LocalCfg
    extends    SomeUserBase
    implements svlibSerializable;
  int    choice;
  string label;
  `SVLIB_DOM_UTILS_BEGIN(LocalCfg)
   `SVLIB_DOM_FIELD_INT(choice)
   `SVLIB_
  `SVLIB_DO
endclass
```

not yet available in all tools

```
interface class svlibSerializable;
  pure virtual function void fromDOM(cfgNodeBase dom);
  pure virtual function cfgNodeBase toDOM();
endclass
```

```
    cfgNodeBase dom);
  if (obj != null)
    obj.fromDOM (dom);
endfunction
```

written by macro

Free Gifts With Every Copy!

- General number reader

```
if (scanVerilogInt("16'h0F_FF", value))
  $display("value = %0d", value);
```

```
value = 4095
```

- Loop over enums

```
typedef {SEVEN=7, NINE=9} num_e;
`foreach_enum(num_e, E, step) begin
  $display("[%0d] %s=%0d", step, E.name, E);
end
```

```
[0] SEVEN=7
[1] NINE=9
```

- Loop over lines in a text file

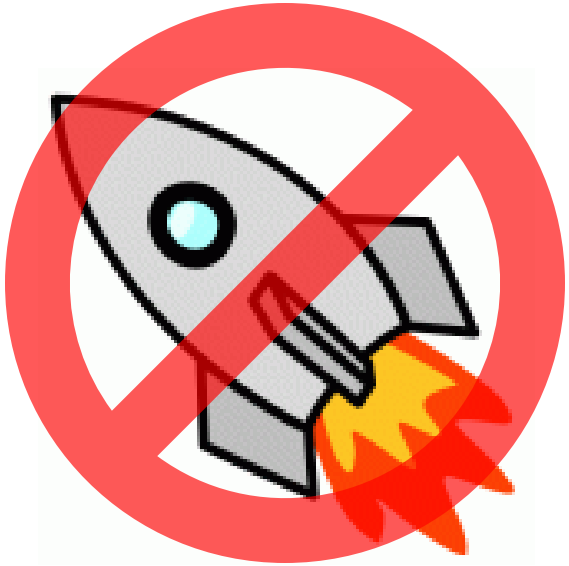
```
int fid = $fopen("hello.txt");
`foreach_line(fid, line, line_number)
  $display("[%0d] %s", line_number, line);
$close(fid);
```

No time to describe…

- Many string manipulation functions
  - slice, insert, append, left/right/center pad
  - find first/last occurrence of substring
- Utility functions wrapper for enums:
  - get length of longest enum name
  - safe conversion from string to enum
  - wildcard match of values against enums that have some X/Z bits
  - queue of all values of enum type
- One easter-egg

great minds think alike, UVM1.2!

Summary



- Nothing super-smart
- Just a bunch of stuff that SV should have had all along

- Making it usable: ***much*** harder than we expected

- Available now
  - beta quality
  - permissive open-source license

Take-away

- Free download at:

www.verilab.com/resources/svlib

- Tell us what you think: **svlib@verilab.com**

- Thanks for listening - any questions?