

使用 **Sulp** 构建你的 **flow**

初衷

1、简化 **flow** ,

单一的 flow

插件式的 flow

瀑布流式的数据传递

2、统一 **sim** 、 **syn** 、 **fpga** 的 **flow** 平台

Verif demo 的基本操作

1、clone 仓库

```
mp $ git clone git@gitlab.cn:sifive.com:project/SFC_BIC.git
```

2、切换到 **sulpur-sharp** 分支

```
mp $ cd SFC_BIC/
```

```
r) $ git checkout sulpur-sharp
```

```
) $ git pull origin sulpur-sharp
```

3、更新 submodule

```
arp) $  
arp) $ git submodule update --init --recursive
```

Verif demo 的基本操作

4、source 环境变量

```
13:32 verify  
$ source sourceme.csh
```

```
harp) $ cd verif
```

```
13:30 env.csh  
$ source env.csh
```

5、生成 sim 使用的 flist

```
$ cd demo/testcase/test/
```

```
after 2 s  
p) $ sulp gen:asicSimFl
```

Verif demo 的基本操作

6、执行仿真

```
after 39 ms  
after 2 s  
irp) $ sulp run:sim
```

7、打开波形

```
) $  
) $ sulp load:wave
```

8、背后的 task

```
1  
2 sulp --tasks  
3  
4 sulp gen:asicSimFl  
5  
6 # sulp build:sccaseApi  
7  
8 # sulp add:sccase --case test  
9 cd testcase/test  
10 # sulp build:sccase  
11  
12 # sulp build:sclib  
13 # sulp run:elab  
14 sulp run:sim  
15 ...  
16  
17 ### verdi  
18 ...  
19 sulp load:wave  
20 ...  
21 ...  
22  
23
```

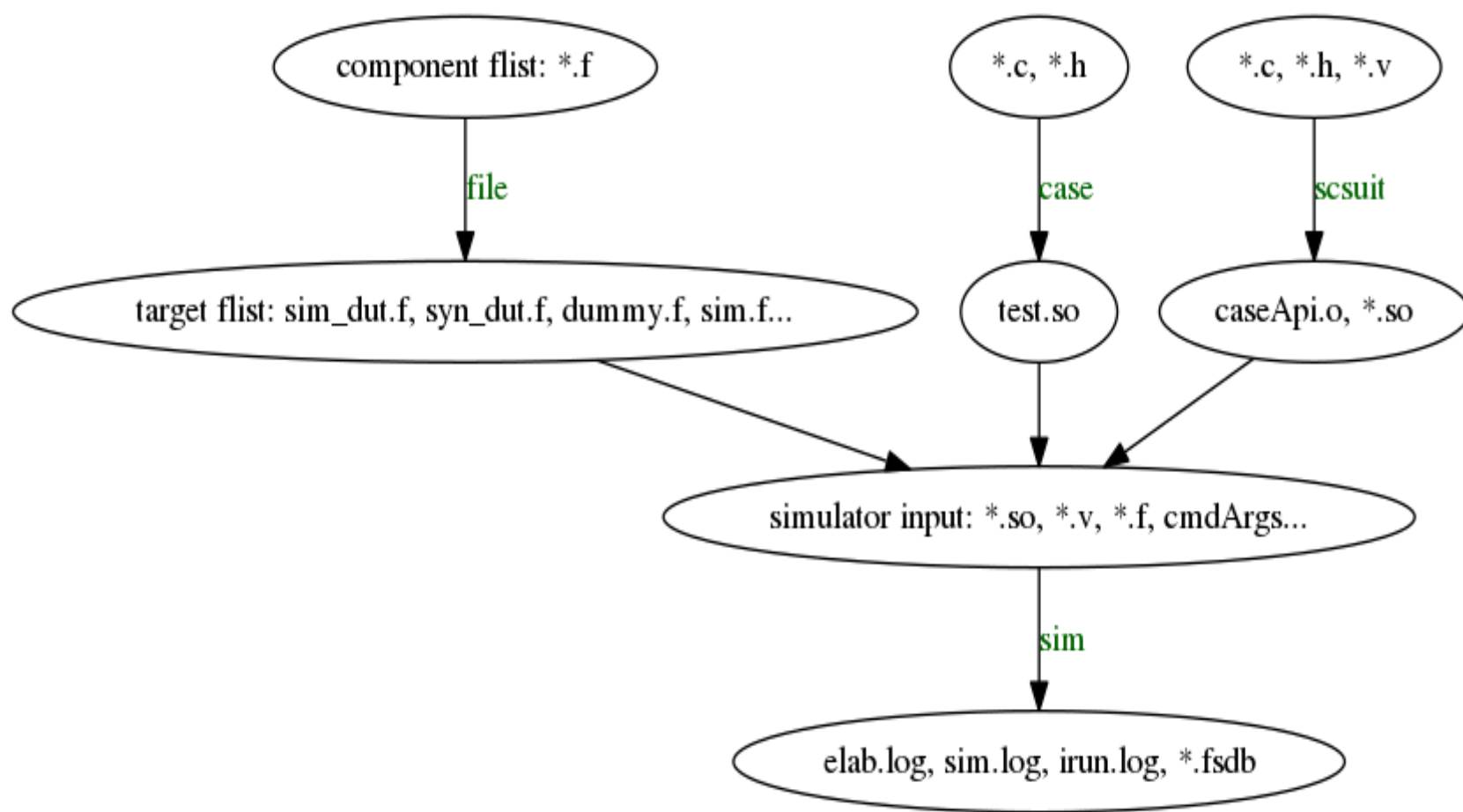
使用 **sulp-verif** 做独立模块的验证

- 1、指定 **flist**
- 2、指定与编译 **testcase**
- 3、编译 **scSuit lib**
- 4、**vlog/elab**
- 5、**sim**

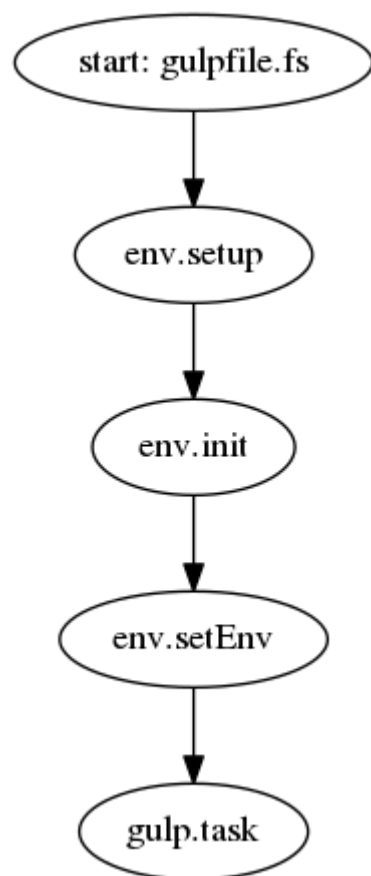
verif 相关的 tasks

```
35] |<series>
35] |   build:scSuitSimLib
35] |   <anonymous>
35] |   run:sim
35] |   <series>
35] |   |   build:sccase
35] |   |   <series>
35] |   |   |   build:scSuitcaseApi
35] |   |   |   <series>
35] |   |   |   |   _linkSCSuitCaseApi
35] |   |   |   |   <series>
35] |   |   |   |   |   _compileSCSuitCaseApi
35] |   |   |   |   |   <anonymous>
35] |   |   |   |   <anonymous>
35] |   |   |   |   _linkSCCase
35] |   |   |   |   <series>
35] |   |   |   |   |   _compileSCCase
35] |   |   |   |   |   <anonymous>
35] |   |   |   |   <anonymous>
35] |   |   |   run:elab
35] |   |   |   <series>
35] |   |   |   |   build:scSuitSimLib
35] |   |   |   |   <anonymous>
35] |   |   |   <anonymous>
35] |   verdi:comp
35] |   verdi:load
35] |   load:wave
```

Verif flow 数据流与设计思路



sulp 执行过程



```
export {};  
require('json5/lib/register')    // isolate the namespace  
const gulp    = require('gulp')  // require the module needed  
....  
import {fooInterface} from "../protocol/FooInterface" // import the flow data structure interface  
  
let flow:fooInterface// declare the flow scope data structure  
let profile:any      // declare the profile variable if needed  
....  
// local function definition  
// gulp task definition  
...  
  
module.exports.flowName = 'foo' // export the flow name, must exist  
  
module.exports.init = ()=>{ // command line arguments definition, must exist  
  program.allowUnknownOption()  
    .option('--option ') // the option can be find in env.getOpt('foo.option')  
  
  module.exports.setEnv = (env)=>{ // fullfill flow data structure, must exist  
    // env is the top structure include  
    // the all flow and global  
    profile=env.useProfile('foo') // instance the profile  
    flow = env.getFlow('foo'); // create the flow data structure instance  
    flow.aaa = env.flow.other.bbb // fullfill the flow data structure  
    flow.bbb = env.getOpt('foo.option')  
  }  
}
```

文件导入，路径处理层，参数注入 :flow/

参数捕获与命令生成执行： profile

开发你的 **flow**

1、基于 **ts** 开发 **gulp** 的 **task**

2、加入已有的脚本