C/UNIX Functions for VHDL Testbenches

(with additional notes on Unix pipes & rsh)

Michael J. Knieser
Rockwell Automation
mjknieser@ra.rockwell.com

Francis G. Wolff
Case Western Reserve University
fxw12@po.cwru.edu

Chris A. Papachristou
Case Western Reserve University
cap@eecs.cwru.edu

Overview

- Motivation
- Issues Implementing C Functions within VHDL
- Common VHDL C Testbench Functions
- Applications of the C Functions (including Unix pipes & rsh)
- Conclusions

Motivation

- Desire to maintain the most portable format for a design
  - "Portability" means that the design is not vendor dependent
  - For hardware… …the portable format is VHDL or Verilog.
  - For software… …the portable format is C.
- Simulation tool foreign interfaces are not the most portable
  - Use of wrappers for ICE & Embedded Testbenches
  - Use of Unix pipes to glue together external EDA tools (including legacy tools)
- Like to keep testbench coding simple...
  - In C… …printf("i=%d\n",i);
  - In VHDL… …write(line,string'("i=")); write(line,i); writeline(output,line);

Issues Implementing C Functions within VHDL

- Pointers
- String Processing
- Passing Variable Number of Arguments
- Passing Different Data Types
- Returning Values

Pointer Issues

- VHDL implements pointer using “ACCESS”; however,...
  - It does not support pointer arithmetic
  - It does not support address referencing
  - due to VHDL strict typing, casting pointer is not supported

- EXAMPLES...

```
char s[10];  …   *p=s+2;    strcpy(p, &s[2]);  strcpy(p,s+2);

        P = (char *)integer_pointer;
```

A classic example of ACCESS is used in TEXTIO:

```
...use  std.textio.all

...type STRING is array (POSITIVE range <>) of CHARACTER;
...type LINE  is access STRING; --pointer to string of characters
...use  std.textio.all;

...variable fprintf_buffer: line;
...
write(printf_buffer, string’(“ALU_OUT=“));
write(printf_buffer, std_logic_vector’(“100UX10”));
writeline(output, printf_buffer);
```

String Processing Issues

- In C a string is an array of character integers
  - `char s[10]; s[1] = s[1] - 'A' + 32;`
- In VHDL a string is an array of enumerated character types
  - `TYPE string IS ARRAY ( POSITIVE RANGE <>) OF character;`
- In C a string is
  - Fixed in size allocation: "`char s[10];`"
  - '`\0`' is used to indicate the termination of a string.
  - But can easily read or write beyond the allocation if a '\0' is not found.
- In VHDL a string is fixed in size and cannot write beyond the limits.
  - `VARIABLE s : string(1 TO 10);`

Passing Variable Number of Arguments

- A C function can use the “varargs.h” library and the ellipsis operator “…” to address any number of arguments
- VHDL supports a fixed number of arguments.

The VHDL workaround is to create the function with the expected largest argument list and utilize VHDL’s default argument assignments.

```
procedure fprintf
( stream          : INOUT   text;
  format          : IN      string;
  a1, a2, a3, a4  : IN      string := “ “;
  a5, a6, a7, a8  : IN      string := “ “ );
```

Other examples:

procedure sprintf(s: INOUT line; format: IN string;

a1, a2, a3, a4, a5, a6, a7, a8 : IN string := " ";

a9, a10, a11, a12, a13, a14, a15, a16: IN string := " ");

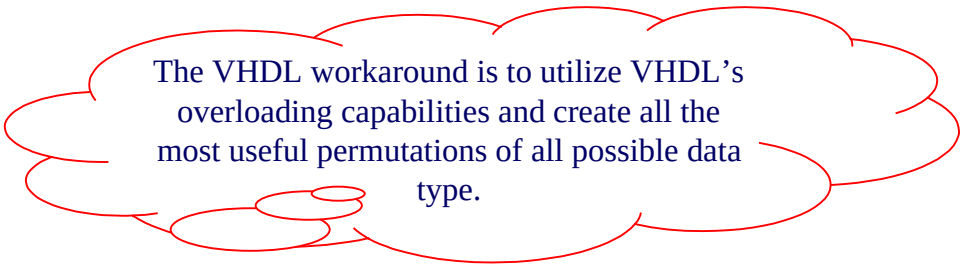
procedure sprintf(s: INOUT string; format: IN string;

a1, a2, a3, a4, a5, a6, a7, a8 : IN string := " ";

a9, a10, a11, a12, a13, a14, a15, a16: IN string := " ");

Passing Different Data Types

- Variable argument data types are not directly recognized syntactically in C at compile time.
  - `printf("%s %d", a, b);`
- VHDL has strict data typing.
  - The VHDL "`printf("%s %d", a, b);`" --a: string; b: integer
  - is a different procedure than "`printf("%d %s", a, b);`".



Other examples of overloading:

function pf(a1: IN integer) return string;

function pf(a1: IN std_ulogic) return string;

function pf(a1: IN std_logic_vector) return string;

procedure printf(format: IN string; a1: string; a2: std_logic);

procedure printf(format: IN string; a1: integer; a2: std_logic);

procedure printf(format: IN string; a1: std_logic; a2: std_logic);

procedure printf(format: IN string; a1: std_logic_vector; a2: std_logic);

procedure sscanf(s: IN string; format: IN string; a1: INOUT std_logic);

procedure sscanf(s: IN string; format: IN string; a1: INOUT std_logic_vector);

Returning Values

- A C function maps to a VHDL function given the following:
  - The C function's caller never ignores the return value: n=atoi(s);
  - The C function's arguments cannot modify original caller's data: strcpy(d, s);
  - Otherwise the C function maps into a VHDL procedure.

- C prototypes

  int fscanf(FILE *stream, const char *format, ...);

  int sscanf(const char *str, const char *format, ...);

- VHDL prototypes

  procedure fscanf(ret: OUT integer; stream: OUT text; format: IN string; ...);

  function sscanf(str: IN string; format: IN string) return integer; --Special case!

Designers would prefer to do:

```
if sscanf(s, "memset %x , %x", address, data)=2 then …
```

than:

```
sscanf(n, s, "memset %x , %x", address, data);
if n=2 then ...
```

Common VHDL C Testbench Functions

- printf, fprintf & sprintf
- scanf, fscanf & sscanf
- Character Stream I/O
- Common String Functions
- Additional Libraries

printf, fprintf & sprintf

- C prototypes
  - `#include <stdio.h>`
  - `int printf (              const char *format, ...);`
  - `int fprintf(FILE *stream, const char *format, ...);`
  - `int sprintf(char *str,    const char *format, ...);`
- VHDL prototypes
  - `LIBRARY   C;`
  - `USE       C.STDIO_H.ALL;`
  - `procedure printf (                  format:IN string; ...);`
  - `procedure fprintf(stream:OUT text;   format:IN string; ...);`
  - `procedure sprintf(str:   OUT string; format:IN string; ...);`
- Examples
  - ```
    variable v:std_logic_vector(15 downto 0);
    printf("ALU_OUT = %20s(%#x)(%o)(%d)\n",v,v,v,v);
    ```
  - ```
    FILE fp: test OPEN WRITE_MODE IS "pipe1";
    fprintf(fp,"ALU_OUT = %u\n",v);
    ```

scanf, fscanf & sscanf

- C prototypes
  - `#include <stdio.h>`
  - `int scanf( const char *format, ...);`
  - `int fscanf(FILE *stream, const char *format, ...);`
  - `int sscanf(const char *str, const char *format, ...);`
- VHDL prototypes
  - `LIBRARY C;`
  - `USE C.STDIO_H.ALL;`
  - `procedure scanf( format:IN string; ...);`
  - `procedure fscanf( stream:OUT text; format:IN string; ...);`
  - `procedure sscanf( str: IN string;format:IN string; ...);`
- Examples
  - `variable v:std_logic_vector(15 downto 0);`
    `scanf("ALU_OUT = %x\n",v);`
  - `FILE fp: test OPEN READ_MODE IS "pipe3";`
    `fscanf(fp,"ALU_OUT = %s\n",v);`

Character Stream I/O

- C prototypes
  - `#include <stdio.h>`
  - `int fputc( int c, FILE *stream);`
  - `int fgetc( FILE *stream);`
- VHDL Example

```
process
  variable c: character;
  FILE     fin:  text OPEN READ_MODE  IS ”pipe2";
  FILE     fout: text OPEN WRITE_MODE IS ”pipe3";
begin
  while not endfile(fin) loop     -- feof(fin) is not allowed...
    fgetc(c, fin);
    if isalpha(c) then fputc(tolower(c), fout);
    else fputc(c, fout); end if;
  end loop;
  fclose(fout); wait;
end process;
```

Single File Stream Out (use default fprintf_buffer);

```
variable v07: std_logic_vector(0 to 7):="01xuhw";
variable v70: std_logic_vector(7 downto 0):="01xuhw";
...
file_open(fout, "data_out.txt", WRITE_MODE);

    fprintf(fout,"%s\n",pf(v07));    --v07=01xuhw
   --warning actual output only occurs whenever a \n is encountered!
   --otherwise the output is held in the shared fprintf_buffer variable.
   --alternative: fprintf(fout, "%s", v07); --depends on library generation

  --fprintf & printf will always output in little endian
    fprintf(fout, "v70=%s", v70\n");  --v70=whux01

  --mixing between fprintf, fputc, fputs, write is allowed
    fputc('#', fout); fputs("end data;" & LF, fout); --"\n" is ignored
    write(fprintf_buffer, v07); --allowed for compatibility

fclose(fout);
```

The fclose(fout) can also be done explicitly fclose(fprintf_buffer, fout);

Example of multi-stream file (explicit buffer) which do not use any shared variables in the stdio_h package

```
variable f1buf, f2buf: line;
...
file_open(f1out, "xxx_out.txt", WRITE_MODE);
file_open(f2out, "xxx_out.txt", WRITE_MODE);

    fprintf(f1buf, f1out, "%s\n", v07); --little endian
    ...
    fprintf(f2buf, f2out, "%s\n", v70\n"); --big endian

fclose(f1buf, f1out);
fclose(f2buf, f2out);
```

Stream I/O (use default buffer fscanf_buffer)

```
file_open(fin, ”data_out.txt", READ_MODE);

  --fscanf can read within a line or multiple text lines
    fscanf(fin, "%3s", s); --read 3 characters max

  --mixing fscanf, fgetc, ungetc, read allowed
  fgetc(s(1), fin); fgetc(s(2), fin); s(5):=NUL;

  --vectors can be read in as strings or as numbers
  --for formats: %x %o %d %u a std_logic 0 is HWXUL0
  --%u is unsigned & %d is signed on msb of std_logic_vector
  fscanf(fin, "%s", v07); fscanf(fin, "%x", v07);
  fscanf(fin, "%u", v07); fscanf(fin, "%d", v07);

file_close(fin);
```

Common String Functions

- VHDL prototypes
  - **LIBRARY C;**
  - **USE C.STRINGS_H.ALL;**
  - **procedure strcpy(dest: OUT string; src: IN string);**
  - **procedure strcpy(dest: INOUT string; di: IN integer; src: IN string);**
  - **procedure strcat(dest: INOUT string; src: IN string);**
  - **procedure strlen(s: IN string; si: IN integer);**

- Examples
  - **variable s, t : string (1 to 256);**
  - **strcpy(s, "hello world");**
  - **strcpy(t, s(8 to 9)); -- array slice supported**
  - **strcat(t, "12345");**
  - **strcpy(t(30 to t'length), "xyzpdq");**
  - **strcpy(t, 30, "xyzpdq"); -- simulating pointer arithmetic**

Code example of string copy:

```
procedure strcpy(d: OUT string; s: IN string) is
   variable dj:integer:=d'left; variable sj:integer:=s'left;
begin
   loop
      if dj>d'right then d(d'right):=NUL; exit; end if;
      if sj>s'right then d(dj):=NUL; exit; end if;
      if s(sj)=NUL  then d(dj):=NUL; exit; end if;
      d(dj):=s(sj); dj:=dj+1; sj:=sj+1;
   end loop;
end strcpy;
```

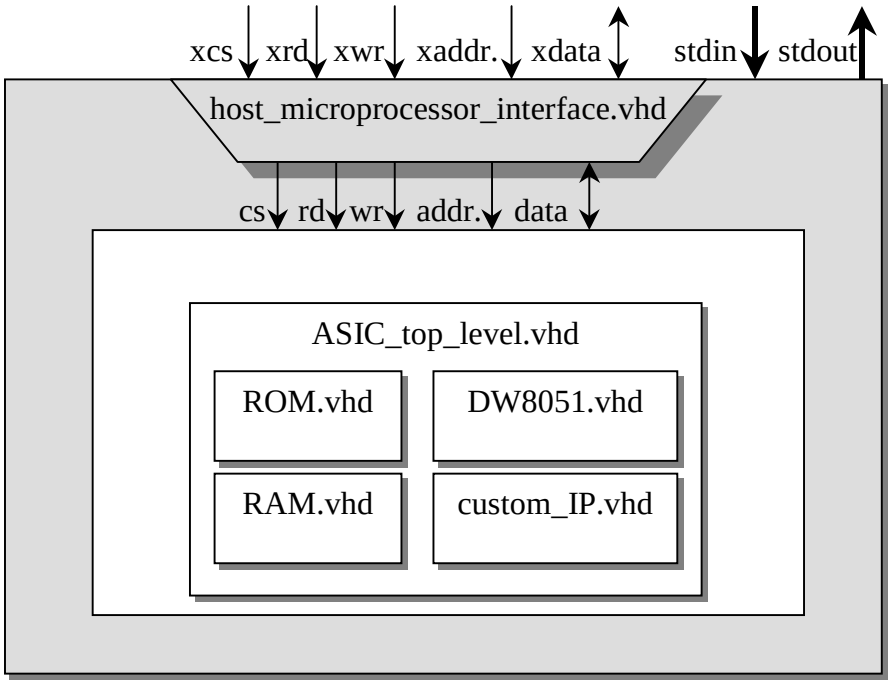
Additional Libraries

- VHDL prototypes
  - `LIBRARY C;`
  - `USE C.CTYPE_H.ALL;`
  - `function isalpha(c: character) return boolean;`
  - `function toupper(c: character) return character;`
  - `USE C.STDLIB_H.ALL;`
  - `function atoi(s: string) return integer;`
  - `USE C.REGEXP_H.ALL;`
  -
```
prodecure regmatch ( ai: OUT integer;   -- alternate match number
                     si: INOUT integer; -- next unmatched character
                     s : IN string;     -- input string
                     f : IN string;     -- PERL pattern matching
                     m1: OUT string;    -- () matching
                     m2: OUT string );  -- () matching
```
- Example
  - `regmatch(ai,fj,fmt,"%([ 0#+-]*)\.?([0-9]*).",m1,m2);`

Note:In order to simplify coding, both fprintf & fscanf internally use regmatch to parse the format control string.

Applications of the C Functions

- Microprocessor Host Testbench



Skeleton Testbench Code with passthru

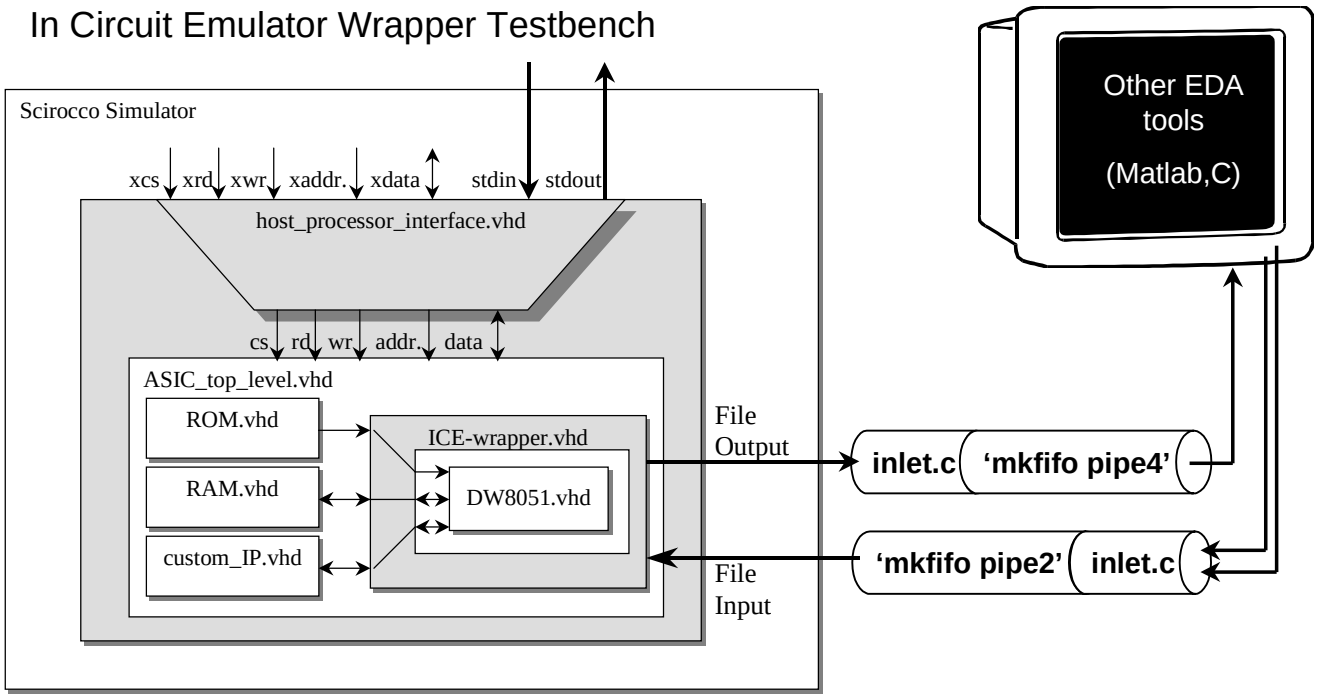
```
loop
    printf("Host => ");
    gets(what_next);
    if ( sscanf(what_next,"write %x %x") = 2 ) then
      sscanf(what_next,"write %x %x",address,data_out);
      wait for CS_START_DELAY;
      cs <= '1'; wait for WR_START_DELAY;
      wr <= '1'; wait for WRITE_WIDTH;
      wr <= '0'; wait for CS_END_DELAY;
      cs <= '0'; wait for WR_END_DELAY;
      address <= "xxxxxxxxxxxxxxx"; data_out <= "ZZZZZZZZ";
    elsif ( sscanf(what_next,"read %x %x") = 2 ) then …
    else           -- passthru wrapper signals
      cs <= xcs;  wr <= xwr;  rd <= xrd;  address <= xaddress;
      data_out <= xdata_out;  xdata_in <= data_in;
    end if;
  end loop;
```

Also, files or pipes can be used:

```
loop
    fprintf(fout, "Host => ");
    fgets(what_next, what_next'length, fin);
    ...
```

Embedded Testbench using UNIX pipes…

- In Circuit Emulator Wrapper Testbench



FIFO pipes: Have the nice property in that they can avoid disk space. Also, The producer generates only as much as the consumer needs. When the consumer cannot absorb the producer pipe, then the producer is I/O blocked until later.

Common example of pipes: `gzip -dc file.tar.gz | tar xvf -`

In this example, the output of gzip is stream directly into the tar program and no additional disk space (or disk access time) is use.

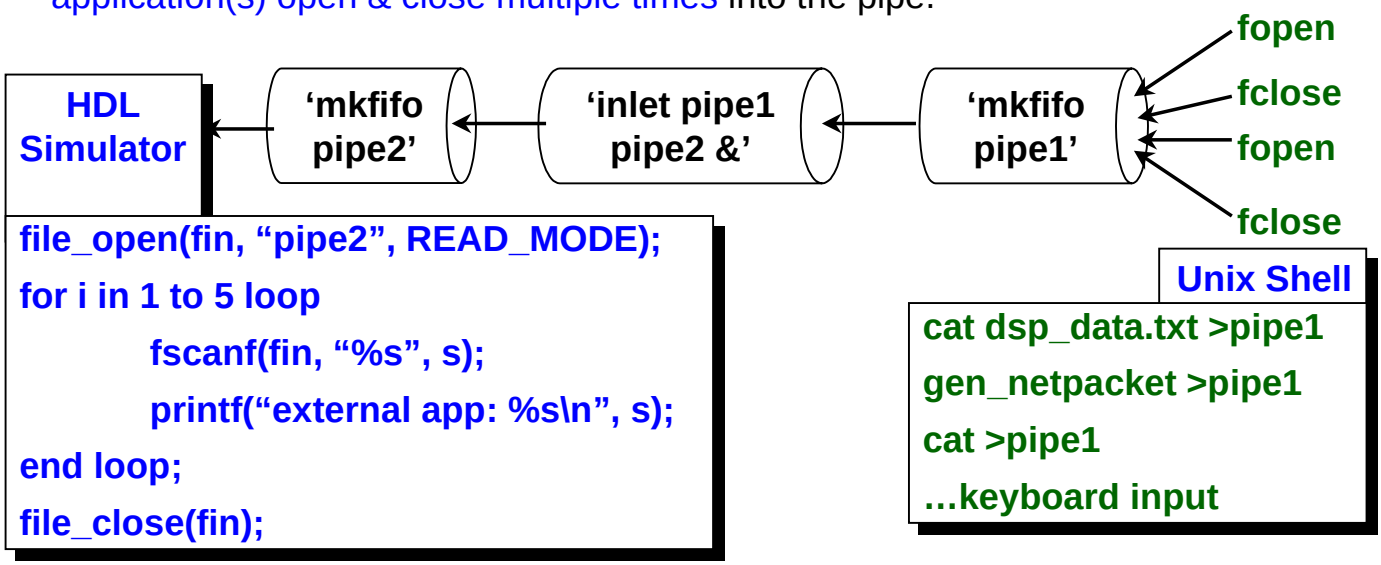
mkfifo (First In First Out) pipes are also know as "named pipes".

"mkfifo pipename" is the same as "mknod pipename p"

For an introduction to named pipes by Andy Vaught, 1997,
see http://www2.linuxjournal.com/lj-issues/issue41/2156.html
or http://www.linuxjournal.com/article.php?sid=2156

inlet pipe

- One difficulty using FIFO pipes is that once an external application issues a file close the pipe also closes and becomes broken.
- The program inlet always keeps the pipe open and let's one or more application(s) open & close multiple times into the pipe.



Use "ps -a" and "kill -9 <process id of inlet>" to terminate the inlet pipe.

The Unix "tee file1 file2 <pipe2 >pipe3" may useful to listen in or create additional pipe outputs.

Even though Network File System, NFS, uses remote procedure call (RPC), it does not support FIFO files (that I know of) across machines.

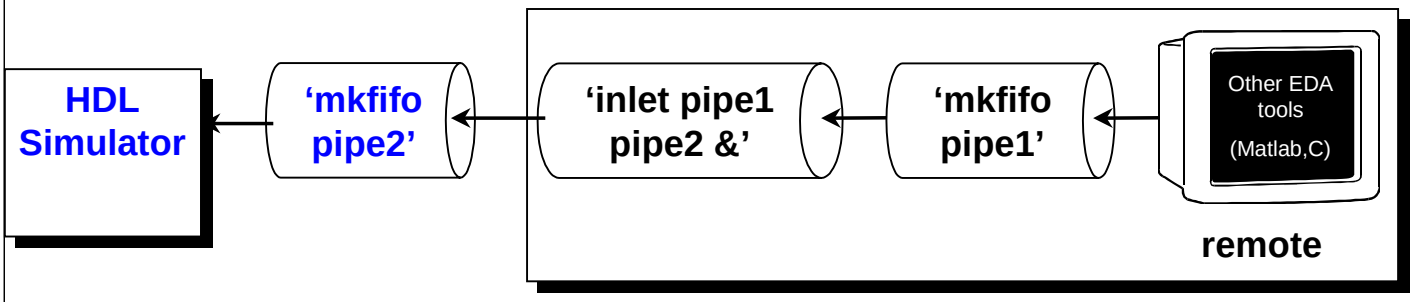
Skeleton inlet.c code

```
fout = open(argv[2], O_WRONLY, 0666);
for(;;) { /* re-open pipe */
       fin = open(argv[1], O_RDONLY, 0);

       while ((len = read(fin, buf, sizeof(buf))) > 0) {
              if (write(fout, buf, len) != len) {
                     perror("inlet: write"); return 1;
              }
       }

       /* len was <= 0; If len = 0, no more data is available.
       Otherwise, an error occurred. */
       if (len < 0) { perror("inlet: read"); return 1; }

       close(fin);
}
```

```
#include <stdio.h> /* compacted version */
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <strings.h>
int main(int argc, char *argv[ ] ) {  int fin, fout; char buf[1024]; int len;
   if (argc == 3) { if (!strcmp(argv[2], "-")) { fout=1; }
    else {  fout = open(argv[2], O_WRONLY, 0666);
      if (fout<0) {  fprintf(stderr, "inlet: open(%s, O_WRONLY, 0666) error\n", argv[2]); exit(1);  }
   }
   for(; ; ) {
    fin = open(argv[1], O_RDONLY, 0);
    if (fin<0) {  fprintf(stderr, "inlet: open(%s, O_RDONLY, 0) error\n", argv[1]); exit(1);  }
    while ((len = read(fin, buf, sizeof(buf))) > 0) {
       if (write(fout, buf, len) != len) {  perror("inlet: write");  return 1;   }
    }
    if (len < 0) { perror("inlet: read"); return 1; }  close(fin);
  } }
  else { fprintf(stderr, "inlet <pipe in> <pipe out>\n  If filename is dash then stdout\n");  }
  return 0;
}
```

rsh pipes

- Need to pipe across machines for simulation parallelism or legacy issues.
- (That legacy EDA tool that can only work on a certain OS, hostid, etc.)



- **`rsh remote mkfifo pipe1`**
- **`mkfifo pipe2`**
- **`(rsh remote -n "cd /home/users/wolff; ./inlet pipe1 -" | cat >pipe2) &`**
- **`# run EDA tool which outputs to pipe1`**
- **`# run HDL simulator which inputs from pipe2`**

We expand on the classic example:

```
tar cf - . | rsh remote "cd /directory; tar xf -"
```

rsh -n flag redirects input to /dev/null. This prevent stdin
from interacting with the console shell terminal.

`( ):` Creates a new shell
`&:` Ampersand starts the remote shell as a background process

Tips for rsh:
(1) Keep the remote ".cshrc" as simple as possible. It basically only needs a path. If necessary create a new user account. Some .cshrc output to stdout which create problems.

(2) Make sure "~/.rhosts", "/etc/hosts.equiv", "/etc/hosts.deny" and "/etc/hosts.allow" are set up correctly.

Conclusions

- Most commonly standard C functions were written.
- These functions simplified the coding of test benches and wrappers with UNIX files, pipes and rsh.
- The web site for this library is located at...

http://bear.ces.cwru.edu/vhdl