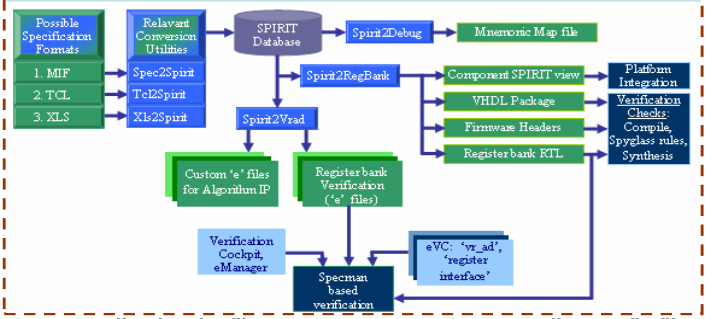
A Methodology for automatic generation of Register bank RTL, related Verification environment and Firmware headers

Saurin Patel, Mukesh Chopra, Bhawna Chopra
STMicroelectronics, Greater Noida, India
{saurin.patel, mukesh.chopra, bhawna.chopra}@st.com

Flow Overview

Goal: To provide a methodology for various projects that should ensure

- Alignment of design specification with corresponding RTL implementation, related verification environment and firmware headers.
- Increased productivity through push button flow to generate thousands of lines of RTL, verification code, 'C, SystemVerilog, Assembly' headers and debugging aids.
- Easy and automated integration of register bank into IP / Subsystem. A typical use model is to assemble the register bank and algorithm block, generated by High Level Synthesis from 'C' code to form an IP / Subsystem.

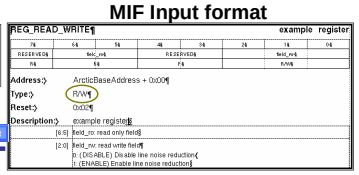


Register bank files generation from Spec

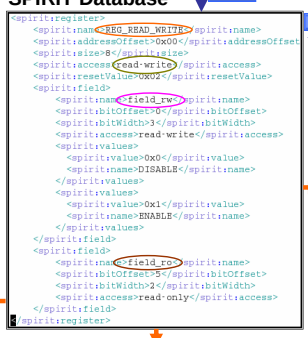
TCL Input format

```
addMemoryMap mem
addAddressBlock  DEFAULT_ADDR 1000 32  0 register rw
addRegister REG_READ_WRITE 0x00 8 rw 0x02 "This is a read write register"
addField field_rw 0 3 rw "read write field"
```

MIF Input format



SPIRIT Database



'C' header file

```
#define REG_READ_WRITE 0x0
```

VHDL package file

'C' hardware defines

```
//  Used for "field_rw" ( @  REG_READ_WRITE)
#define                DISABLE          0x0
#define                ENABLE           0x1
```

SystemVerilog header file

```
define ip_A_REGS_BASE 32'hFC0B4000

/* Register definitions */

/* This is a read write register */
define ip_A_REG_READ_WRITE                  (`ip_A_REGS_BASE+32'h0000_0000)
define ip_A_REG_READ_WRITE(x)               (`ip_A_REGS_BASE + 'h00000000 + ((x)*'h4000))

// Definitions for the bits in the  REG_READ_WRITE register//
define ip_field_rw_VALUE(x)                 ((x)&'h07)<<0)
define ip_field_rw_MSB                      (2)
define ip_field_rw_LSB                      (0)
define ip_field_rw_MASK                     ('h07)
define ip_field_rw                          ((`ip_field_rw_MASK) << (`ip_field_rw_LSB))
```

SystemVerilog config file

```
set type verilog

# Modules
set modules {
        {ip.xml                 0xFC0B4000 0x4000}
}
```

Spirit2Header

Assembly config file

```
set type assembly

# Modules
set modules {
        {ip.xml                 0xFC0B4000 0x4000}
}
```

Spirit2Header

Assembly header file

```
.equ ip_A_REGS_BASE,                0xFC0B4000

## Register definitions

# This is a read write registe
.equ ip_A_REG_READ_WRITE                    (ip_A_REGS_BASE+0x00000000)

 ## BIT LEVEL DEFINITIONS AND MACROS

## Definitions for the bits in the  REG_READ_WRITE register
.equ ip_field_rw_MSB,               (2)
.equ ip_field_rw_LSB,               (0)
.equ ip_field_rw_MASK,              (0x07)
.equ ip_field_rw,                   ((ip_field_rw_MASK) << (ip_field_rw_LSB))
```

Mnemonic Map (.svcf)

```
nmap new -reuse -name {New map}
-contents {
{%x=0000 -bgcolor #ff9900 -font
{-*-courier-bold-r-normal--12-*-
-label {REG_READ_WRITE} -linecolor
white -shape bus}
}
```

Spirit2Debug

.TRN File | .DSN File

Verilog fragment of Register slice

```
always @(posedge clock or negedge reset_n) begin
  if (~reset_n) begin
  //Reset states of Registers
    field_rw[2:0] <= 3'h2;
  end
  else begin

    //T1 Write section
    if (~opc[0] && req && (~r_req_temp) && (~opc[3])) begin
      case (addr[addr_width-1:0])
        `REG_READ_WRITE + T1_REG_BASE_ADDRESS : begin
          if (be[0]) begin
            if (!i_write_protection_disable) begin
              field_rw[2:0] <= data[2:0];
            end
          end
        end
      endcase
    end
    //T1 Read section
    else if (opc[0] && req && (~r_req_temp) && (~opc[3])) begin
      case (addr[addr_width-1:0])
        `REG_READ_WRITE + T1_REG_BASE_ADDRESS : begin
          if (be[0]) begin
            r_data[2:0] <= field_rw[2:0];
            r_data[6:5] <= field_ro[1:0];
          end
        end
      endcase
    end
  end
end
```

SPIRIT view of Register bank

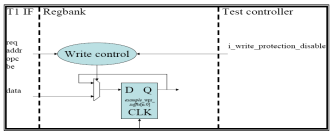
```
<spirit:signal>
  <spirit:name>field_rw</spirit:name>
  <spirit:direction>out</spirit:direction>
  <spirit:left>2</spirit:left>
  <spirit:right>0</spirit:right>
</spirit:signal>
<spirit:signal>
  <spirit:name>i_write_protection_disable</spirit:name>
  <spirit:direction>in</spirit:direction>
</spirit:signal>
<spirit:signal>
  <spirit:name>field_ro</spirit:name>
  <spirit:direction>in</spirit:direction>
  <spirit:left>1</spirit:left>
  <spirit:right>0</spirit:right>
</spirit:signal>
```

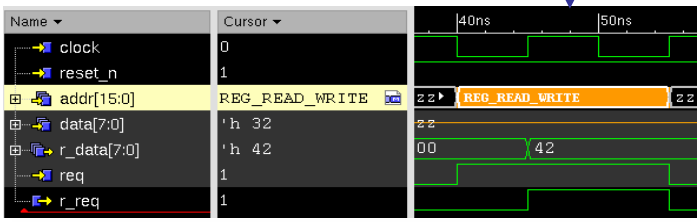
Configuration Input format

Spirit2RegBank

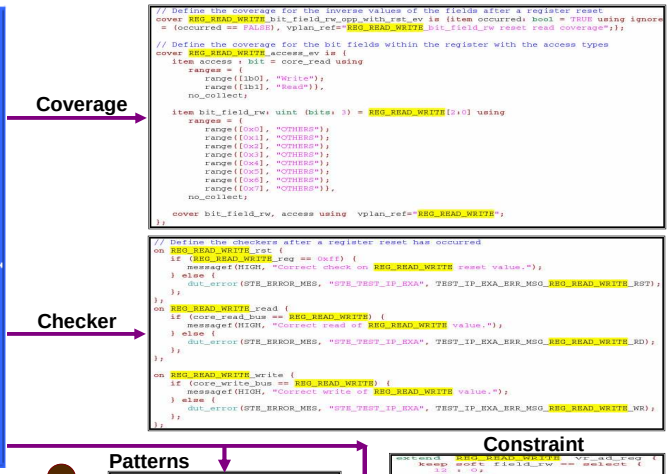
```
[WPR]
wr_peri  =  _rw
```

Special Register Behavior

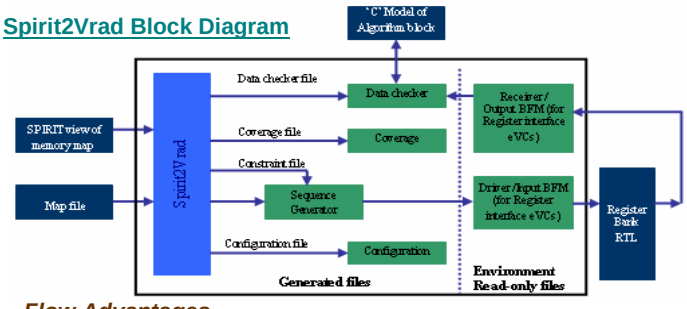




SPIRIT to Verification files



Spirit2Vrad Block Diagram



Flow Advantages

- **Increased Reliability** by reducing risk of errors as output is always aligned with specification
- **Improved cycle time** by reducing costly loops between firmware & hardware teams at a later stage
- **Productivity gain** by having push button flow to reduce development costs and save time for higher value added tasks
- **Standardization** of the flow is based on IPXACT (IEEE P1685) standard, given by SPIRIT
- **Higher Reuse** by using common inputs for RTL, verification, header and configuration generation

Results

- Usage in more than **50 IPs**
- Demonstrated flow on many ***(~10)*** taped out chips
- An example of **11000 lines** of RTL code is generated in **2 minutes** which could be reduced further

Future Scope

- **Special behavior:** As IPXACT **1.5** supports special registers, reduce use of additional configuration input.
- **Low Power:** Think about concept of low power register bank.