GRAND CHALLENGE - 2025

IP Porting with Shakti SoC

DEVELOPED BY :
SHAKTI DEVELOPMENT TEAM @ IITM
SHAKTI.ORG.IN

Introduction

Three variants are there to port peripheral IPs with Shakti.

Steps involved in porting peripheral IPs with Shakti.

- Do the following in soc.defines.
    - Add one more slave number for the new IP .
    - Add a memory map for the same.
- Do the following in the soc.bsv or cluster file(uart_cluster, pwm cluster, etc.) based on the implementation.
    - Make one instance of the IP interface to be ported.
    - Check for the memory map of the IP and set the slave number for the new IP.
    - Connect AXI4(lite) of the core with peripheral's AXI4(lite).
    - If necessary take the interface to the top file (fpga_top.v).
    - Take the interrupt pins to the top level or PLIC based on the implementation.
- Do the following in the fpga_top.v
    - Declare the IO pads for the peripheral.
    - Map them to the mkSoc.
    - If the pin(s) are bidirectional, connect them through an IO buffer.
- Do the following in the constraints.xdc.
    - Add signal to pin mapping for all the IO pads.

Integrating Slow peripherals with Shakti:

Slow peripherals operate @ lower frequencies from few Hz to few MHz (upto 25MHz). They can be integrated with Shakti core using the AXI4 Lite interface which is slow fabric.

The peripherals like SPI, UART, I2C, GPIO, etc. are interfaced with Shakti using the following logic. In addition, it is assumed that the peripheral IP is already integrated with Shakti. The objective is to add one more additional interface of the same type.

Normally these peripherals are interfaced with Core through clusters (e.g. UART Cluster, SPI cluster, PWM cluster, Mixed cluster, etc.). This section explains how to add the existing IPs to the core.
It is assumed that the shakti core is taken from sp2020 gitlab repo (https://gitlab.com/shaktiproject/sp2020).

Hierarchy

For UART the hierarchy is as follows (leftmost is the top file),

Soc ← uart_cluster

```
package Soc;
  (*synthesize*)
    let curr_reset<-exposeCurrentReset;
    Ifc_ccore_axi4 ccore <- mkccore_axi4(`resetpc, 0);

    AXI4_Fabric_IFC #(`Num_Fast_Masters, `Num_Fast_Slaves, `paddr, ELEN, USERSPACE)
                                                                  fabric <- mkAXI4_Fabric
                                                                  (fn_slave_map_fast);
    Ifc_clint_axi4#(`paddr, ELEN, 0, 1, 256) clint <- mkclint_axi4();
    Ifc_debug_halt_loop_axi4#(`paddr, ELEN, USERSPACE) debug_memory <-
    mkdebug_halt_loop_axi4;
    Ifc_err_slave_axi4#(`paddr,ELEN,0) fast_err_slave <- mkerr_slave_axi4;

    AXI4_Lite_Fabric_IFC #(`Num_Masters, `Num_Slaves, `paddr, 32, USERSPACE)
                                                                   slow_fabric <- mkAXI4_Lite_Fabric
                                                                   (fn_slave_map);
    Ifc_pwm_cluster pwm_cluster <- mkpwm_cluster;
    Ifc_uart_cluster uart_cluster <- mkuart_cluster;
    Ifc_spi_cluster spi_cluster <- mkspi_cluster;
    Ifc_mixed_cluster mixed_cluster <- mkmixed_cluster;
    Ifc_err_slave_axi4lite#(`paddr,32,0) err_slave <- mkerr_slave_axi4lite;
    Ifc_bram_axi4lite#(`paddr, 32, 0,  13) boot <- mkbram_axi4lite('h1000, "boot.mem",
    "Boot");
    Wire#(Bit#(3)) wr_ext_interrupts <- mkWire();

    Wire#(Bit#(1)) wr_gpio4_in <- mkDWire(0);
    Wire#(Bit#(1)) wr_gpio7_in <- mkDWire(0);
    Wire#(Bit#(1)) wr_gpio8_in <- mkDWire(0);
    Wire#(Bit#(1)) wr_gpio14_in <- mkDWire(0);
    Wire#(Bit#(1)) wr_gpio15_in <- mkDWire(0);
```

All the Memory mapping details will be in the Soc.defines file.

```
`define FastErr_slave_num 3
`define Slow_fabric_slave_num 4

`define MemoryBase        'h8000_0000
`define MemoryEnd         'h8FFF_FFFF
`define ClintBase         'h0200_0000
`define ClintEnd          'h020B_FFFF
`define DebugBase         'h0000_0010
`define DebugEnd          'h0000_001F
`define SlowBase          'h0000_1000
`define SlowEnd           'h0004_7FFF

`define Num_Masters 1

`define Num_Slaves 7
`define PWMCluster_slave_num 0
`define UARTCluster_slave_num 1
`define SPICluster_slave_num 2
`define MixedCluster_slave_num 3
`define Boot_slave_num 4
`define Eth_slave_num 5
`define Err_slave_num 6

`define BootBase          'h0000_1000
`define BootEnd           'h0000_2FFF
`define UARTClusterBase   'h0001_1300
`define UARTClusterEnd    'h0001_1540
`define SPIClusterBase    'h0002_0000
`define SPIClusterEnd     'h0002_01FF
`define PWMClusterBase    'h0003_0000
```

Memory Mapping

All the slow peripherals will be placed in the range of Slow fabric memory allocation. UART peripheral also will fall in this range.

```
`define Num_Fast_Slaves 5
`define Memory_slave_num 0
`define Clint_slave_num 1
`define Debug_slave_num 2
`define FastErr_slave_num 3
`define Slow_fabric_slave_num 4

`define MemoryBase        'h8000_0000
`define MemoryEnd         'h8FFF_FFFF
`define ClintBase         'h0200_0000
`define ClintEnd          'h020B_FFFF
`define DebugBase         'h0000_0010
`define DebugEnd          'h0000_001F
`define SlowBase          'h0000_1000
`define SlowEnd           'h0004_7FFF

`define Num_Masters 1

`define Num_Slaves 7
`define PWMCluster_slave_num 0
`define UARTCluster_slave_num 1
`define SPICluster_slave_num 2
`define MixedCluster_slave_num 3
`define Boot_slave_num 4
`define Eth_slave_num 5
`define Err_slave_num 6

`define BootBase          'h0000_1000
`define BootEnd           'h0000_2FFF
```

In this file the Slow fabric memory allocation has further separate memory range allocated for UART instances alone which is termed as UART cluster, all the memory mapping of the UART instances will be in this range.

Slow fabric → UART cluster → UART instance

Same like UART clusters there are also clusters for SPI and PWM to accommodate SPI & PWM IPs, all the remaining peripherals will be placed in the Mixed cluster.

Adding one more cluster IP:

This section shows the integration for one more UART IP in the UART cluster.
The files that needs to be modified are

- soc.bsv
- soc.defines
- uart_cluster.bsv
- mixed_cluster.bsv
- fpga_top.v

- constraints.xdc

1. Make the following changes in the “soc.defines”.

Step 1 (Increase the UART Cluster memory range):

Increase the UART Cluster memory map from

```
`define BootBase          'h0000_1000
`define BootEnd           'h0000_2FFF
`define UARTClusterBase   'h0001_1300
`define UARTClusterEnd    'h0001_1540
`define SPIClusterBase    'h0002_0000
`define SPIClusterEnd     'h0002_01FF
`define PWMClusterBase    'h0003_0000
`define PWMClusterEnd     'h0003_05FF
`define MixedClusterBase  'h0004_0000
`define MixedClusterEnd   'h0004_15FF
`define EthBase           'h0004_4000
`define EthEnd            'h0004_7FFF

// - PWM cluster
`define PWMCluster_Num_Slaves 7
`define PWMCluster_Num_Masters 1

`define PWM0_slave_num 0
`define PWM1_slave_num 1
`define PWM2_slave_num 2
`define PWM3_slave_num 3
`define PWM4_slave_num 4
`define PWM5_slave_num 5

`define PWMCluster_err_slave_num 6

`define PWM0Base     'h0003_0000
`define PWM0End      'h0003_00FF
`define PWM1Base     'h0003_0100
`define PWM1End      'h0003_01FF
```

to

```
`define Err_slave_num 6

`define BootBase          'h0000_1000
`define BootEnd           'h0000_2FFF
`define UARTClusterBase   'h0001_1300
`define UARTClusterEnd    'h0001_1640
`define SPIClusterBase    'h0002_0000
`define SPIClusterEnd     'h0002_01FF
`define PWMClusterBase    'h0003_0000
`define PWMClusterEnd     'h0003_05FF
`define MixedClusterBase  'h0004_0000
`define MixedClusterEnd   'h0004_15FF
`define EthBase           'h0004_4000
`define EthEnd            'h0004_7FFF

// - PWM cluster
`define PWMCluster_Num_Slaves 7
`define PWMCluster_Num_Masters 1

`define PWM0_slave_num 0
`define PWM1_slave_num 1
`define PWM2_slave_num 2
`define PWM3_slave_num 3
`define PWM4_slave_num 4
`define PWM5_slave_num 5

`define PWMCluster_err_slave_num 6

`define PWM0Base     'h0003_0000
`define PWM0End      'h0003_00FF
```

Step 2 (Increase the number of UART slaves):
Increase the “UARTCluster_Num_Slaves” from 4 to 5.

```
79  `define SPIcluster_err_slave_num 2
80
81  `define SPI0Base     'h0002_0000
82  `define SPI0End      'h0002_00FF
83  `define SPI1Base     'h0002_0100
84  `define SPI1End      'h0002_01FF
85  //`define SPI2Base     'h0002_0200
86  //`define SPI2End      'h0002_02FF
87
88  // - UART cluster
89  `define UARTCluster_Num_Slaves 4
90  `define UARTCluster_Num_Masters 1
91
92  `define UART0_slave_num 0
93  `define UART1_slave_num 1
94  `define UART2_slave_num 2
95  `define UARTCluster_err_slave_num 3
96
97  `define UART0Base     'h0001_1300
98  `define UART0End      'h0001_1340
99  `define UART1Base     'h0001_1400
100 `define UART1End      'h0001_1440
101 `define UART2Base     'h0001_1500
102 `define UART2End      'h0001_1540
103
104 // - Mixed Cluster
105 `define MixedCluster_Num_Slaves 7
106 `define MixedCluster_Num_Masters 1
107
108 `define I2C0_slave_num 0
109 `define I2C1_slave_num 1
```

From “4” to “5”.

```
`define SPICluster_err_slave_num 2

`define SPI0Base     'h0002_0000
`define SPI0End      'h0002_00FF
`define SPI1Base     'h0002_0100
`define SPI1End      'h0002_01FF
//`define SPI2Base     'h0002_0200
//`define SPI2End      'h0002_02FF

// - UART cluster
`define UARTCluster_Num_Slaves 5
`define UARTCluster_Num_Masters 1

`define UART0_slave_num 0
`define UART1_slave_num 1
`define UART2_slave_num 2
`define UARTCluster_err_slave_num 3

`define UART0Base     'h0001_1300
`define UART0End      'h0001_1340
`define UART1Base     'h0001_1400
`define UART1End      'h0001_1440
`define UART2Base     'h0001_1500
`define UART2End      'h0001_1540

// - Mixed Cluster
`define MixedCluster_Num_Slaves 7
`define MixedCluster_Num_Masters 1

`define I2C0_slave_num 0
`define I2C1_slave_num 1
```

Step 3 (Add slave number for the new UART IP):
Add a slave number for the new slave UART. i.e. Change the

```
79  `define SPICluster_err_slave_num 2
80
81  `define SPI0Base     'h0002_0000
82  `define SPI0End      'h0002_00FF
83  `define SPI1Base     'h0002_0100
84  `define SPI1End      'h0002_01FF
85  //`define SPI2Base     'h0002_0200
86  //`define SPI2End      'h0002_02FF
87
88  // - UART cluster
89  `define UARTCluster_Num_Slaves 4
90  `define UARTCluster_Num_Masters 1
91
92  `define UART0_slave_num 0
93  `define UART1_slave_num 1
94  `define UART2_slave_num 2
95  `define UARTCluster_err_slave_num 3
96
97  `define UART0Base     'h0001_1300
98  `define UART0End      'h0001_1340
99  `define UART1Base     'h0001_1400
100 `define UART1End      'h0001_1440
101 `define UART2Base     'h0001_1500
102 `define UART2End      'h0001_1540
103
104 // - Mixed Cluster
105 `define MixedCluster_Num_Slaves 7
106 `define MixedCluster_Num_Masters 1
107
108 `define I2C0_slave_num 0
109 `define I2C1_slave_num 1
```

to

```
`define SPI0Base      'h0002_0000
`define SPI0End       'h0002_00FF
`define SPI1Base      'h0002_0100
`define SPI1End       'h0002_01FF
//`define SPI2Base      'h0002_0200
//`define SPI2End       'h0002_02FF

// - UART cluster
`define UARTCluster_Num_Slaves 5
`define UARTCluster_Num_Masters 1

`define UART0_slave_num 0
`define UART1_slave_num 1
`define UART2_slave_num 2
`define UART3_slave_num 3
`define UARTCluster_err_slave_num 4

`define UART0Base     'h0001_1300
`define UART0End      'h0001_1340
`define UART1Base     'h0001_1400
`define UART1End      'h0001_1440
`define UART2Base     'h0001_1500
`define UART2End      'h0001_1540
`define UART3Base     'h0001_1600
`define UART3End      'h0001_1640

// - Mixed Cluster
`define MixedCluster_Num_Slaves 7
`define MixedCluster_Num_Masters 1
```

Step 4 (Add memory range for the new UART being added):
Add a memory map to the new UART.

```
`define SPIcluster_err_slave_num 2

`define SPI0Base      'h0002_0000
`define SPI0End       'h0002_00FF
`define SPI1Base      'h0002_0100
`define SPI1End       'h0002_01FF
//`define SPI2Base      'h0002_0200
//`define SPI2End       'h0002_02FF

// - UART cluster
`define UARTCluster_Num_Slaves 5
`define UARTCluster_Num_Masters 1

`define UART0_slave_num 0
`define UART1_slave_num 1
`define UART2_slave_num 2
`define UART2_slave_num 3
`define UARTCluster_err_slave_num 4

`define UART0Base     'h0001_1300
`define UART0End      'h0001_1340
`define UART1Base     'h0001_1400
`define UART1End      'h0001_1440
`define UART2Base     'h0001_1500
`define UART2End      'h0001_1540

// - Mixed Cluster
`define MixedCluster_Num_Slaves 7
`define MixedCluster_Num_Masters 1

`define I2C0_slave_num 0
```

i.e . Add memory map

```
`define SPIcluster_err_slave_num 2

`define SPI0Base      'h0002_0000
`define SPI0End       'h0002_00FF
`define SPI1Base      'h0002_0100
`define SPI1End       'h0002_01FF
//`define SPI2Base      'h0002_0200
//`define SPI2End       'h0002_02FF

// - UART cluster
`define UARTCluster_Num_Slaves 5
`define UARTCluster_Num_Masters 1

`define UART0_slave_num 0
`define UART1_slave_num 1
`define UART2_slave_num 2
`define UART2_slave_num 3
`define UARTCluster_err_slave_num 4

`define UART0Base     'h0001_1300
`define UART0End      'h0001_1340
`define UART1Base     'h0001_1400
`define UART1End      'h0001_1440
`define UART2Base     'h0001_1500
`define UART2End      'h0001_1540
`define UART3Base     'h0001_1600
`define UART3End      'h0001_1640

// - Mixed Cluster
`define MixedCluster_Num_Slaves 7
`define MixedCluster_Num_Masters 1
```

2. Make the following changes in the “uart_cluster.bsv”

Step 1 (Add condition for memory range check):

Add if condition for new uart i.e. change the code from

```
package uart_cluster;
  import AXI4_Lite_Fabric::*;
  import uart::*;
  import err_slave::*;
  import Connectable:: *;
  import GetPut:: *;

  `include "Soc.defines"

  function Bit#(TLog#(`UARTCluster_Num_Slaves)) fn_slave_map (Bit#(`paddr) addr);
    Bool slave_exist = True;
    Bit#(TLog#(`UARTCluster_Num_Slaves)) slave_num = 0;
    if(     addr>=  `UART0Base  &&  addr<=  `UART0End  )
      slave_num  =  `UART0_slave_num;
    else  if(addr>=  `UART1Base  &&  addr<=  `UART1End  )
      slave_num  =  `UART1_slave_num;
    else  if(addr>=  `UART2Base  &&  addr<=  `UART2End  )
      slave_num  =  `UART2_slave_num;
    else
      slave_num  =  `UARTCluster_err_slave_num;
    
    return slave_num;
  endfunction:fn_slave_map

  interface Ifc_uart_cluster;
    interface RS232 uart0_io;
    interface RS232 uart1_io;
    interface RS232 uart2_io;
    interface AXI4_Lite_Slave_IFC#(`paddr, 32, 0) slave;
    method Bit#(3) uart_interrupts;
  endinterface
```

to

```
package uart_cluster;
  import AXI4_Lite_Fabric::*;
  import uart::*;
  import err_slave::*;
  import Connectable:: *;
  import GetPut:: *;

  `include "Soc.defines"

  function Bit#(TLog#(`UARTCluster_Num_Slaves)) fn_slave_map (Bit#(`paddr) addr);
    Bool slave_exist = True;
    Bit#(TLog#(`UARTCluster_Num_Slaves)) slave_num = 0;
    if(     addr>= `UART0Base && addr<= `UART0End )
      slave_num =  `UART0_slave_num;
    else if(addr>= `UART1Base && addr<= `UART1End )
      slave_num =  `UART1_slave_num;
    else if(addr>= `UART2Base && addr<= `UART2End )
      slave_num =  `UART2_slave_num;
    else if(addr>= `UART3Base && addr<= `UART3End )
      slave_num =  `UART3_slave_num;
    else
      slave_num = `UARTCluster_err_slave_num;

    return slave_num;
  endfunction:fn_slave_map

  interface Ifc_uart_cluster;
    interface RS232 uart0_io;
    interface RS232 uart1_io;
    interface RS232 uart2_io;
    interface AXI4_Lite_Slave_IFC#(`paddr, 32, 0) slave;
```

Step 2 (Make one more instance for new UART):
Add UART Interface i.e. Change the code from

```
Welcome | Soc.bsv | uart_cluster.bsv ● | Soc.defines | Settings
uart_cluster.bsv > {} uart_cluster > Ifc_uart_cluster
uart0 Aa ab .* ? of 11
31  package uart_cluster;
41    function Bit#(TLog#(`UARTCluster_Num_Slaves)) fn_slave_map (Bit#(`paddr) addr);
55      return slave_num;
56    endfunction:fn_slave_map
57
58    interface Ifc_uart_cluster;
59      interface RS232 uart0_io;
60      interface RS232 uart1_io;
61      interface RS232 uart2_io;
62      interface AXI4_Lite_Slave_IFC#(`paddr, 32, 0) slave;
63      method Bit#(3) uart_interrupts;
64    endinterface
65
66    (*synthesize*)
67    module mkuart(Ifc_uart_axi4lite#(32, 32, 0, 16));
68      let core_clock<-exposeCurrentClock;
69      let core_reset<-exposeCurrentReset;
70      let ifc();
71      mkuart_axi4lite#(core_clock, core_reset, 163, 0, 0) _temp(ifc);
72      return ifc;
73    endmodule
74
75    (*synthesize*)
76    module mkuart_cluster(Ifc_uart_cluster);
77      let curr_clk<- exposeCurrentClock;
78      let curr_reset <- exposeCurrentReset;
79      AXI4_Lite_Master_Xactor_IFC #(`paddr, 32, 0) c2m_xactor <- mkAXI4_Lite_Master_Xactor;
80      AXI4_Lite_Slave_Xactor_IFC #(`paddr, 32, 0) c2s_xactor <- mkAXI4_Lite_Slave_Xactor;
81      AXI4_Lite_Fabric_IFC #(`UARTCluster_Num_Masters, `UARTCluster_Num_Slaves, `paddr, 32,
        0)
```

to

```
Welcome | Soc.bsv | uart_cluster.bsv ● | Soc.defines | Settings
uart_cluster.bsv > {} uart_cluster > Ifc_uart_cluster
uart0 Aa ab .* 4 of 11
31  package uart_cluster;
41    function Bit#(TLog#(`UARTCluster_Num_Slaves)) fn_slave_map (Bit#(`paddr) addr);
55      return slave_num;
56    endfunction:fn_slave_map
57
58    interface Ifc_uart_cluster;
59      interface RS232 uart0_io;
60      interface RS232 uart1_io;
61      interface RS232 uart2_io;
62      interface RS232 uart3_io;
63      interface AXI4_Lite_Slave_IFC#(`paddr, 32, 0) slave;
64      method Bit#(3) uart_interrupts;
65    endinterface
66
67    (*synthesize*)
68    module mkuart(Ifc_uart_axi4lite#(32, 32, 0, 16));
69      let core_clock<-exposeCurrentClock;
70      let core_reset<-exposeCurrentReset;
71      let ifc();
72      mkuart_axi4lite#(core_clock, core_reset, 163, 0, 0) _temp(ifc);
73      return ifc;
74    endmodule
75
76    (*synthesize*)
77    module mkuart_cluster(Ifc_uart_cluster);
78      let curr_clk<- exposeCurrentClock;
79      let curr_reset <- exposeCurrentReset;
80      AXI4_Lite_Master_Xactor_IFC #(`paddr, 32, 0) c2m_xactor <- mkAXI4_Lite_Master_Xactor;
81      AXI4_Lite_Slave_Xactor_IFC #(`paddr, 32, 0) c2s_xactor <- mkAXI4_Lite_Slave_Xactor;
82      AXI4_Lite_Fabric_IFC #(`UARTCluster_Num_Masters, `UARTCluster_Num_Slaves, `paddr, 32,
```

Step 3 (Add one more bit to take interrupt to Soc.bsv):

Add one more bit for the new UART Interrupt i.e. change the code from **Bit#(3)**

```
Welcome | Soc.bsv | uart_cluster.bsv ● | Soc.defines | Settings
uart_cluster.bsv > {} uart_cluster > Ifc_uart_cluster
uart0  Aa  ab  .*  4 of 11

31  package uart_cluster;
41    function Bit#(TLog#(`UARTCluster_Num_Slaves)) fn_slave_map (Bit#(`paddr) addr);
55      return slave_num;
56    endfunction:fn_slave_map
57
58    interface Ifc_uart_cluster;
59      interface RS232 uart0_io;
60      interface RS232 uart1_io;
61      interface RS232 uart2_io;
62      interface RS232 uart3_io;
63      interface AXI4_Lite_Slave_IFC#(`paddr, 32, 0) slave;
64      method Bit#(3) uart_interrupts;
65    endinterface
66
67    (*synthesize*)
68    module mkuart(Ifc_uart_axi4lite#(32, 32, 0, 16));
69      let core_clock<-exposeCurrentClock;
70      let core_reset<-exposeCurrentReset;
71      let ifc();
72      mkuart_axi4lite#(core_clock, core_reset, 163, 0, 0) _temp(ifc);
73      return ifc;
74    endmodule
75
76    (*synthesize*)
77    module mkuart_cluster(Ifc_uart_cluster);
78      let curr_clk<- exposeCurrentClock;
79      let curr_reset <- exposeCurrentReset;
80      AXI4_Lite_Master_Xactor_IFC #(`paddr, 32, 0) c2m_xactor <- mkAXI4_Lite_Master_Xactor;
81      AXI4_Lite_Slave_Xactor_IFC #(`paddr, 32, 0) c2s_xactor <- mkAXI4_Lite_Slave_Xactor;
82      AXI4_Lite_Fabric_IFC #(`UARTCluster_Num_Masters, `UARTCluster_Num_Slaves, `paddr, 32,
```

to Bit#(4)

```
31  package uart_cluster;
41    function Bit#(TLog#(`UARTCluster_Num_Slaves)) fn_slave_map (Bit#(`paddr) addr);
55      return slave_num;
56    endfunction:fn_slave_map
57
58    interface Ifc_uart_cluster;
59      interface RS232 uart0_io;
60      interface RS232 uart1_io;
61      interface RS232 uart2_io;
62      interface RS232 uart3_io;
63      interface AXI4_Lite_Slave_IFC#(`paddr, 32, 0) slave;
64      method Bit#(4) uart_interrupts;
65    endinterface
66
67    (*synthesize*)
68    module mkuart(Ifc_uart_axi4lite#(32, 32, 0, 16));
69      let core_clock<-exposeCurrentClock;
70      let core_reset<-exposeCurrentReset;
71      let ifc();
72      mkuart_axi4lite#(core_clock, core_reset, 163, 0, 0) _temp(ifc);
73      return ifc;
74    endmodule
75
76    (*synthesize*)
77    module mkuart_cluster(Ifc_uart_cluster);
78      let curr_clk<- exposeCurrentClock;
79      let curr_reset <- exposeCurrentReset;
80      AXI4_Lite_Master_Xactor_IFC #(`paddr, 32, 0) c2m_xactor <- mkAXI4_Lite_Master_Xactor;
81      AXI4_Lite_Slave_Xactor_IFC #(`paddr, 32, 0) c2s_xactor <- mkAXI4_Lite_Slave_Xactor;
82      AXI4_Lite_Fabric_IFC #(`UARTCluster_Num_Masters, `UARTCluster_Num_Slaves, `paddr, 32,
```

Setp 4 (Add a UART module instance by assigning it to a new interface instance):
Add one more instance for the UART interface i.e. change the code from

```
Welcome | Soc.bsv | uart_cluster.bsv ● | Soc.defines | Settings
uart_cluster.bsv > {} uart_cluster > {} mkuart_cluster
31  package uart_cluster;
67    (*synthesize*)
69      let core_clock<-exposeCurrentClock;
70      let core_reset<-exposeCurrentReset;
71      let ifc();
72      mkuart_axi4lite#(core_clock, core_reset, 163, 0, 0) _temp(ifc);
73      return ifc;
74    endmodule
75
76    (*synthesize*)
77    module mkuart_cluster(Ifc_uart_cluster);
78      let curr_clk<- exposeCurrentClock;
79      let curr_reset <- exposeCurrentReset;
80      AXI4_Lite_Master_Xactor_IFC #(`paddr, 32, 0) c2m_xactor <- mkAXI4_Lite_Master_Xactor;
81      AXI4_Lite_Slave_Xactor_IFC #(`paddr, 32, 0) c2s_xactor <- mkAXI4_Lite_Slave_Xactor;
82      AXI4_Lite_Fabric_IFC #(`UARTCluster_Num_Masters, `UARTCluster_Num_Slaves, `paddr, 32,0)
83                                                                  fabric <- mkAXI4_Lite_Fabric(fn_slave_map);
84      let uart0 <- mkuart();
85      let uart1 <- mkuart();
86      let uart2 <- mkuart();
87      Ifc_err_slave_axi4lite#(`paddr, 32, 0 ) err_slave <- mkerr_slave_axi4lite;
88
89      mkConnection(c2m_xactor.axi_side, fabric.v_from_masters[0]);
90
91      mkConnection(c2s_xactor.o_wr_addr,c2m_xactor.i_wr_addr);
92      mkConnection(c2s_xactor.o_wr_data,c2m_xactor.i_wr_data);
93      mkConnection(c2m_xactor.o_wr_resp,c2s_xactor.i_wr_resp);
94      mkConnection(c2s_xactor.o_rd_addr,c2m_xactor.i_rd_addr);
95      mkConnection(c2m_xactor.o_rd_data,c2s_xactor.i_rd_data);
96
```

to

```
Welcome | Soc.bsv | uart_cluster.bsv ● | Soc.defines | Settings
uart_cluster.bsv > {} uart_cluster > {} mkuart_cluster
31  package uart_cluster;
67    (*synthesize*)
69      let core_clock<-exposeCurrentClock;
70      let core_reset<-exposeCurrentReset;
71      let ifc();
72      mkuart_axi4lite#(core_clock, core_reset, 163, 0, 0) _temp(ifc);
73      return ifc;
74    endmodule
75
76    (*synthesize*)
77    module mkuart_cluster(Ifc_uart_cluster);
78      let curr_clk<- exposeCurrentClock;
79      let curr_reset <- exposeCurrentReset;
80      AXI4_Lite_Master_Xactor_IFC #(`paddr, 32, 0) c2m_xactor <- mkAXI4_Lite_Master_Xactor;
81      AXI4_Lite_Slave_Xactor_IFC #(`paddr, 32, 0) c2s_xactor <- mkAXI4_Lite_Slave_Xactor;
82      AXI4_Lite_Fabric_IFC #(`UARTCluster_Num_Masters, `UARTCluster_Num_Slaves, `paddr, 32,0)
83                                                                  fabric <- mkAXI4_Lite_Fabric(fn_slave_map);
84      let uart0 <- mkuart();
85      let uart1 <- mkuart();
86      let uart2 <- mkuart();
87      let uart3 <- mkuart();
88      Ifc_err_slave_axi4lite#(`paddr, 32, 0 ) err_slave <- mkerr_slave_axi4lite;
89
90      mkConnection(c2m_xactor.axi_side, fabric.v_from_masters[0]);
91
92      mkConnection(c2s_xactor.o_wr_addr,c2m_xactor.i_wr_addr);
93      mkConnection(c2s_xactor.o_wr_data,c2m_xactor.i_wr_data);
94      mkConnection(c2m_xactor.o_wr_resp,c2s_xactor.i_wr_resp);
95      mkConnection(c2s_xactor.o_rd_addr,c2m_xactor.i_rd_addr);
96      mkConnection(c2m_xactor.o_rd_data,c2s_xactor.i_rd_data);
```

Step 5 (Connect the AXI4 interface with new UART):

Make a connection to the AXI4 interface i.e. Change the code from

```
Welcome | Soc.bsv | uart_cluster.bsv ● | Soc.defines | Settings
uart_cluster.bsv > {} uart_cluster > {} mkuart_cluster
31  package uart_cluster;
76    (*synthesize*)
84      let uart0 <- mkuart();
85      let uart1 <- mkuart();
86      let uart2 <- mkuart();
87      let uart3 <- mkuart();
88      Ifc_err_slave_axi4lite#(`paddr, 32, 0 ) err_slave <- mkerr_slave_axi4lite;
89
90      mkConnection(c2m_xactor.axi_side, fabric.v_from_masters[0]);
91
92      mkConnection(c2s_xactor.o_wr_addr,c2m_xactor.i_wr_addr);
93      mkConnection(c2s_xactor.o_wr_data,c2m_xactor.i_wr_data);
94      mkConnection(c2m_xactor.o_wr_resp,c2s_xactor.i_wr_resp);
95      mkConnection(c2s_xactor.o_rd_addr,c2m_xactor.i_rd_addr);
96      mkConnection(c2m_xactor.o_rd_data,c2s_xactor.i_rd_data);
97                                                          uart0
98  ·→··mkConnection·(fabric.v_to_slaves·[`UART0_slave_num·],uart0.slave);
99  ·→··mkConnection·(fabric.v_to_slaves·[`UART1_slave_num·],uart1.slave);
100 ·→··mkConnection·(fabric.v_to_slaves·[`UART2_slave_num·],uart2.slave);
101     mkConnection (fabric.v_to_slaves [`UARTCluster_err_slave_num ] , err_slave.slave);
102
103     interface uart0_io=uart0.io;
104     interface uart1_io=uart1.io;
105     interface uart2_io=uart2.io;
106     interface slave= c2s_xactor.axi_side;
107     method Bit#(3) uart_interrupts;
108       return {uart2.interrupt, uart1.interrupt, uart0.interrupt};
109       //return {uart2.interrupt, uart1.interrupt, uart0.interrupt};
110     endmethod
111   endmodule
```

to

```
Welcome | Soc.bsv | uart_cluster.bsv ● | Soc.defines | Settings
uart_cluster.bsv > {} uart_cluster > {} mkuart_cluster
31  package uart_cluster;
76    (*synthesize*)
84      let uart0 <- mkuart();
85      let uart1 <- mkuart();
86      let uart2 <- mkuart();
87      let uart3 <- mkuart();
88      Ifc_err_slave_axi4lite#(`paddr, 32, 0 ) err_slave <- mkerr_slave_axi4lite;
89
90      mkConnection(c2m_xactor.axi_side, fabric.v_from_masters[0]);
91
92      mkConnection(c2s_xactor.o_wr_addr,c2m_xactor.i_wr_addr);
93      mkConnection(c2s_xactor.o_wr_data,c2m_xactor.i_wr_data);
94      mkConnection(c2m_xactor.o_wr_resp,c2s_xactor.i_wr_resp);
95      mkConnection(c2s_xactor.o_rd_addr,c2m_xactor.i_rd_addr);
96      mkConnection(c2m_xactor.o_rd_data,c2s_xactor.i_rd_data);
97
98      mkConnection (fabric.v_to_slaves [`UART0_slave_num ],uart0.slave);
99      mkConnection (fabric.v_to_slaves [`UART1_slave_num ],uart1.slave);
100     mkConnection (fabric.v_to_slaves [`UART2_slave_num ],uart2.slave);
101     mkConnection (fabric.v_to_slaves [`UART3_slave_num ],uart3.slave);
102     mkConnection (fabric.v_to_slaves [`UARTCluster_err_slave_num ] , err_slave.slave);
103
104     interface uart0_io=uart0.io;
105     interface uart1_io=uart1.io;
106     interface uart2_io=uart2.io;
107     interface slave= c2s_xactor.axi_side;
108     method Bit#(3) uart_interrupts;
109       return {uart2.interrupt, uart1.interrupt, uart0.interrupt};
110       //return {uart2.interrupt, uart1.interrupt, uart0.interrupt};
111     endmethod
```

Step 6 (Add a new UART interface instance to take the UART interface from uart cluster into fpga_top.v):

Add a new UART interface to soc.bsv by adding the following i.e. change the code from

```
Welcome | Soc.bsv | uart_cluster.bsv ● | Soc.defines | Settings
uart_cluster.bsv > {} uart_cluster > {} mkuart_cluster
31  package uart_cluster;
76    (*synthesize*)
84      let uart0 <- mkuart();
85      let uart1 <- mkuart();
86      let uart2 <- mkuart();
87      let uart3 <- mkuart();
88      Ifc_err_slave_axi4lite#(`paddr, 32, 0 ) err_slave <- mkerr_slave_axi4lite;
89
90      mkConnection(c2m_xactor.axi_side, fabric.v_from_masters[0]);
91
92      mkConnection(c2s_xactor.o_wr_addr,c2m_xactor.i_wr_addr);
93      mkConnection(c2s_xactor.o_wr_data,c2m_xactor.i_wr_data);
94      mkConnection(c2m_xactor.o_wr_resp,c2s_xactor.i_wr_resp);
95      mkConnection(c2s_xactor.o_rd_addr,c2m_xactor.i_rd_addr);
96      mkConnection(c2m_xactor.o_rd_data,c2s_xactor.i_rd_data);
97
98      mkConnection (fabric.v_to_slaves [`UART0_slave_num ],uart0.slave);
99      mkConnection (fabric.v_to_slaves [`UART1_slave_num ],uart1.slave);
100     mkConnection (fabric.v_to_slaves [`UART2_slave_num ],uart2.slave);
101     mkConnection (fabric.v_to_slaves [`UART3_slave_num ],uart3.slave);
102     mkConnection (fabric.v_to_slaves [`UARTCluster_err_slave_num ] , err_slave.slave);
103
104     ····interface·uart0_io=uart0.io;
105     ····interface·uart1_io=uart1.io;
106     ····interface·uart2_io=uart2.io;
107     interface slave= c2s_xactor.axi_side;
108     method Bit#(3) uart_interrupts;
109       return {uart2.interrupt, uart1.interrupt, uart0.interrupt};
110       //return {uart2.interrupt, uart1.interrupt, uart0.interrupt};
111     endmethod
```

to

```
Welcome | Soc.bsv | uart_cluster.bsv ● | Soc.defines | Settings
uart_cluster.bsv > {} uart_cluster > {} mkuart_cluster
31  package uart_cluster;
76    (*synthesize*)
84      let uart0 <- mkuart();
85      let uart1 <- mkuart();
86      let uart2 <- mkuart();
87      let uart3 <- mkuart();
88      Ifc_err_slave_axi4lite#(`paddr, 32, 0 ) err_slave <- mkerr_slave_axi4lite;
89
90      mkConnection(c2m_xactor.axi_side, fabric.v_from_masters[0]);
91
92      mkConnection(c2s_xactor.o_wr_addr,c2m_xactor.i_wr_addr);
93      mkConnection(c2s_xactor.o_wr_data,c2m_xactor.i_wr_data);
94      mkConnection(c2m_xactor.o_wr_resp,c2s_xactor.i_wr_resp);
95      mkConnection(c2s_xactor.o_rd_addr,c2m_xactor.i_rd_addr);
96      mkConnection(c2m_xactor.o_rd_data,c2s_xactor.i_rd_data);
97
98      mkConnection (fabric.v_to_slaves [`UART0_slave_num ],uart0.slave);
99      mkConnection (fabric.v_to_slaves [`UART1_slave_num ],uart1.slave);
100     mkConnection (fabric.v_to_slaves [`UART2_slave_num ],uart2.slave);
101     mkConnection (fabric.v_to_slaves [`UART3_slave_num ],uart3.slave);
102     mkConnection (fabric.v_to_slaves [`UARTCluster_err_slave_num ] , err_slave.slave);
103
104     interface uart0_io=uart0.io;
105     interface uart1_io=uart1.io;
106     interface uart2_io=uart2.io;
107     ····interface·uart3_io=uart3.io;
108
109     interface slave= c2s_xactor.axi_side;
110     method Bit#(3) uart_interrupts;
111       return {uart2.interrupt, uart1.interrupt, uart0.interrupt};
```

Step 7 (Add one more UART Interrupt bit to take the same to PLIC):

Add an interrupt bit for the new UART i.e. change the code from

```
Welcome | Soc.bsv | uart_cluster.bsv ● | Soc.defines | Settings
uart_cluster.bsv > {} uart_cluster > {} mkuart_cluster > uart_interrupts
31  package uart_cluster;
76    (*synthesize*)
86      let uart2 <- mkuart();
87      let uart3 <- mkuart();
88      Ifc_err_slave_axi4lite#(`paddr, 32, 0 ) err_slave <- mkerr_slave_axi4lite;
89
90      mkConnection(c2m_xactor.axi_side, fabric.v_from_masters[0]);
91
92      mkConnection(c2s_xactor.o_wr_addr,c2m_xactor.i_wr_addr);
93      mkConnection(c2s_xactor.o_wr_data,c2m_xactor.i_wr_data);
94      mkConnection(c2m_xactor.o_wr_resp,c2s_xactor.i_wr_resp);
95      mkConnection(c2s_xactor.o_rd_addr,c2m_xactor.i_rd_addr);
96      mkConnection(c2m_xactor.o_rd_data,c2s_xactor.i_rd_data);
97
98      mkConnection (fabric.v_to_slaves [`UART0_slave_num ],uart0.slave);
99      mkConnection (fabric.v_to_slaves [`UART1_slave_num ],uart1.slave);
100     mkConnection (fabric.v_to_slaves [`UART2_slave_num ],uart2.slave);
101     mkConnection (fabric.v_to_slaves [`UART3_slave_num ],uart3.slave);
102     mkConnection (fabric.v_to_slaves [`UARTCluster_err_slave_num ] , err_slave.slave);
103
104     interface uart0_io=uart0.io;
105     interface uart1_io=uart1.io;
106     interface uart2_io=uart2.io;
107     interface uart3_io=uart3.io;
108
109     interface slave= c2s_xactor.axi_side;
110     method Bit#(3) uart_interrupts;        interrupt
111 → → → return·{uart2.interrupt,·uart1.interrupt,·uart0.interrupt};
112       //return {uart2.interrupt, uart1.interrupt, uart0.interrupt};
113     endmethod
```

To

```
    let uart0 <- mkuart();
    let uart1 <- mkuart();
    let uart2 <- mkuart();
    let uart3 <- mkuart();
    Ifc_err_slave_axi4lite#(`paddr, 32, 0 ) err_slave <- mkerr_slave_axi4lite;

    mkConnection(c2m_xactor.axi_side, fabric.v_from_masters[0]);

    mkConnection(c2s_xactor.o_wr_addr,c2m_xactor.i_wr_addr);
    mkConnection(c2s_xactor.o_wr_data,c2m_xactor.i_wr_data);
    mkConnection(c2m_xactor.o_wr_resp,c2s_xactor.i_wr_resp);
    mkConnection(c2s_xactor.o_rd_addr,c2m_xactor.i_rd_addr);
    mkConnection(c2m_xactor.o_rd_data,c2s_xactor.i_rd_data);

    mkConnection (fabric.v_to_slaves [`UART0_slave_num ],uart0.slave);
    mkConnection (fabric.v_to_slaves [`UART1_slave_num ],uart1.slave);
    mkConnection (fabric.v_to_slaves [`UART2_slave_num ],uart2.slave);
    mkConnection (fabric.v_to_slaves [`UART2_slave_num ],uart3.slave);
    mkConnection (fabric.v_to_slaves [`UARTCluster_err_slave_num ] , err_slave.slave);

    interface uart0_io=uart0.io;
    interface uart1_io=uart1.io;
    interface uart2_io=uart2.io;
    interface uart3_io=uart3.io;
    interface slave= c2s_xactor.axi_side;
    method Bit#(4) uart_interrupts;
        return {uart3.interrupt, uart2.interrupt, uart1.interrupt, uart0.interrupt};
    endmethod
  endmodule
endpackage
```

3. Make the following changes in the “soc.bsv”.

Step 1(Add one more interface instance for UART to take connect new UART in the UART cluster to top level file (fpga_top.v):

Make the top interface for “fpga_top.v” i.e add the UART3 interface (UART1 & UART2 are pin muxed so they are taken care separately) by changing the code from

```
Welcome | Soc.bsv | uart_cluster.bsv | Soc.defines | Settings
Soc.bsv > {} Soc > Ifc_Soc
31  package Soc;
85    function Bit#(TLog#(`Num_Slaves)) fn_slave_map (Bit#(`paddr) addr);
92      else if(addr >= `SPIClusterBase && addr <= `SPIClusterEnd)
93        slave_num = `SPICluster_slave_num;
94      else if(addr >= `MixedClusterBase && addr <= `MixedClusterEnd)
95        slave_num = `MixedCluster_slave_num;
96      else if(addr >= `PLICBase && addr <= `PLICEnd)
97        slave_num = `MixedCluster_slave_num;
98      else if(addr >= `BootBase && addr <= `BootEnd)
99        slave_num = `Boot_slave_num;
100     else if (addr >= `EthBase && addr <= `EthEnd)
101       slave_num = `Eth_slave_num;
102     else
103       slave_num = `Err_slave_num;
104
105     return slave_num;
106   endfunction:fn_slave_map
107
108   interface Ifc_Soc;
109     interface Ifc_spi_io spi0_io;
110 //    interface Ifc_spi_io spi2_io;
111     interface RS232 uart0_io;
112     method I2C_out i2c0_out;                    //I2c IO interface
113     method I2C_out i2c1_out;                    //I2c IO interface
114     (*always_ready, always_enabled*)
115     interface AXI4_Lite_Master_IFC#(`paddr, 32, 0) xadc_master;
116     interface AXI4_Lite_Master_IFC#(`paddr, 32, 0) eth_master;
117     interface AXI4_Master_IFC#(`paddr, ELEN, 0) mem_master;
118     interface IOCellSide iocell_io;
119     (*always_enabled,always_ready*)
```

to

```
Welcome | Soc.bsv | uart_cluster.bsv | Soc.defines | Settings
Soc.bsv > {} Soc > Ifc_Soc
31  package Soc;
85    function Bit#(TLog#(`Num_Slaves)) fn_slave_map (Bit#(`paddr) addr);
92      else if(addr >= `SPIClusterBase && addr <= `SPIClusterEnd)
93        slave_num = `SPICluster_slave_num;
94      else if(addr >= `MixedClusterBase && addr <= `MixedClusterEnd)
95        slave_num = `MixedCluster_slave_num;
96      else if(addr >= `PLICBase && addr <= `PLICEnd)
97        slave_num = `MixedCluster_slave_num;
98      else if(addr >= `BootBase && addr <= `BootEnd)
99        slave_num = `Boot_slave_num;
100     else if (addr >= `EthBase && addr <= `EthEnd)
101       slave_num = `Eth_slave_num;
102     else
103       slave_num = `Err_slave_num;
104
105     return slave_num;
106   endfunction:fn_slave_map
107
108   interface Ifc_Soc;
109     interface Ifc_spi_io spi0_io;
110 //    interface Ifc_spi_io spi2_io;
111     interface RS232 uart0_io;
112     interface RS232 uart3_io;
113     method I2C_out i2c0_out;                    //I2c IO interface
114     method I2C_out i2c1_out;                    //I2c IO interface
115     (*always_ready, always_enabled*)
116     interface AXI4_Lite_Master_IFC#(`paddr, 32, 0) xadc_master;
117     interface AXI4_Lite_Master_IFC#(`paddr, 32, 0) eth_master;
118     interface AXI4_Master_IFC#(`paddr, ELEN, 0) mem_master;
119     interface IOCellSide iocell_io;
120     (*always_enabled,always_ready*)
```

Step 2 (Assign the new interface to UART cluster interface):

Assign the interface with UART cluster by changing the code from

```
Welcome | Soc.bsv | uart_cluster.bsv | Soc.defines | Settings
Soc.bsv > {} Soc > {} mkSoc
 31  package Soc;
275    (*synthesize*)
631      method gpio_19_outen = mixed_cluster.gpio_io.gpio_out_en[19];
632      method gpio_20_outen = mixed_cluster.gpio_io.gpio_out_en[20];
633      method gpio_21_outen = mixed_cluster.gpio_io.gpio_out_en[21];
634      method gpio_22_outen = mixed_cluster.gpio_io.gpio_out_en[22];
635      method gpio_23_outen = mixed_cluster.gpio_io.gpio_out_en[23];
636      method gpio_24_outen = mixed_cluster.gpio_io.gpio_out_en[24];
637      method gpio_25_outen = mixed_cluster.gpio_io.gpio_out_en[25];
638      method gpio_26_outen = mixed_cluster.gpio_io.gpio_out_en[26];
639      method gpio_27_outen = mixed_cluster.gpio_io.gpio_out_en[27];
640      method gpio_28_outen = mixed_cluster.gpio_io.gpio_out_en[28];
641      method gpio_29_outen = mixed_cluster.gpio_io.gpio_out_en[29];
642      method gpio_30_outen = mixed_cluster.gpio_io.gpio_out_en[30];
643      method gpio_31_outen = mixed_cluster.gpio_io.gpio_out_en[31];
644      interface spi0_io = spi_cluster.spi0_io;
645  //    interface spi1_io = spi_cluster.   uart0_io
646  ····interface·uart0_io·=·uart_cluster.uart0_io;
647      method  i2c0_out = mixed_cluster.i2c0_out;                  //I2c IO interface
648      method  i2c1_out = mixed_cluster.i2c1_out;                  //I2c IO interface
649      interface iocell_io = mixed_cluster.pinmuxtop_iocell_side;            //GPIO IO interface
650      interface xadc_master = mixed_cluster.xadc_master;
651      interface eth_master = slow_fabric.v_to_slaves[`Eth_slave_num];
652      interface mem_master = fabric.v_to_slaves [`Memory_slave_num];
653      method Action ext_interrupts(Bit#(3) i);
654        wr_ext_interrupts <= i;
655      endmethod
656    `ifdef rtldump
657      interface io_dump= eclass.io_dump;
658    `endif
```

to

```
Welcome | Soc.bsv | uart_cluster.bsv | Soc.defines | Settings
Soc.bsv > {} Soc > {} mkSoc
 31  package Soc;
275    (*synthesize*)
631      method gpio_19_outen = mixed_cluster.gpio_io.gpio_out_en[19];
632      method gpio_20_outen = mixed_cluster.gpio_io.gpio_out_en[20];
633      method gpio_21_outen = mixed_cluster.gpio_io.gpio_out_en[21];
634      method gpio_22_outen = mixed_cluster.gpio_io.gpio_out_en[22];
635      method gpio_23_outen = mixed_cluster.gpio_io.gpio_out_en[23];
636      method gpio_24_outen = mixed_cluster.gpio_io.gpio_out_en[24];
637      method gpio_25_outen = mixed_cluster.gpio_io.gpio_out_en[25];
638      method gpio_26_outen = mixed_cluster.gpio_io.gpio_out_en[26];
639      method gpio_27_outen = mixed_cluster.gpio_io.gpio_out_en[27];
640      method gpio_28_outen = mixed_cluster.gpio_io.gpio_out_en[28];
641      method gpio_29_outen = mixed_cluster.gpio_io.gpio_out_en[29];
642      method gpio_30_outen = mixed_cluster.gpio_io.gpio_out_en[30];
643      method gpio_31_outen = mixed_cluster.gpio_io.gpio_out_en[31];
644      interface spi0_io = spi_cluster.spi0_io;
645  //    interface spi1_io = spi_cluster.spi1_io;
646  ····interface·uart0_io·=·uart_cluster.uart0_io;
647  ····interface·uart3_io·=·uart_cluster.uart3_io;
648      method  i2c0_out = mixed_cluster.i2c0_out;                  //I2c IO interface
649      method  i2c1_out = mixed_cluster.i2c1_out;                  //I2c IO interface
650      interface iocell_io = mixed_cluster.pinmuxtop_iocell_side;            //GPIO IO interface
651      interface xadc_master = mixed_cluster.xadc_master;
652      interface eth_master = slow_fabric.v_to_slaves[`Eth_slave_num];
653      interface mem_master = fabric.v_to_slaves [`Memory_slave_num];
654      method Action ext_interrupts(Bit#(3) i);
655        wr_ext_interrupts <= i;
656      endmethod
657    `ifdef rtldump
658      interface io_dump= eclass.io_dump;
```

4. Make the following changes in the “mixed_cluster.bsv”.

Step 1 (add one more interrupt for new uart):

Do the following changes in mixed_cluster.bsv i.e. increase the interrupt bits from 12 to 13. Change the code from

```
Explorer (Ctrl+Shift+E) - 2 unsaved files   mixed_cluster.bsv   uart_cluster.bsv ●   Soc.defines   Settings
mixed_cluster.bsv > {} mixed_cluster > Ifc_mixed_cluster
31  package mixed_cluster;
41    import plic :: * ;
42    import pinmux :: * ;
43    import pinmux_axi4lite :: * ;
44    `include "Soc.defines"
45
46    interface Ifc_mixed_cluster;
47      method I2C_out i2c0_out;                    //I2c IO interface
48      method I2C_out i2c1_out;                    //I2c IO interface
49      method Bit#(1) sb_ext_interrupt;
50      (*always_ready, always_enabled*)
51      interface GPIO#(32) gpio_io;              //GPIO IO interface
52      interface IOCellSide pinmuxtop_iocell_side;
53      interface PeripheralSide pinmuxtop_peripheral_side;
54      (*always_ready, always_enabled*)
55      method Action interrupts(Bit#(12) inp);
56      interface AXI4_Lite_Slave_IFC#(`paddr, 32, 0) slave;
57      interface AXI4_Lite_Master_IFC#(`paddr, 32, 0) xadc_master;
58    endinterface
59
60    (*synthesize*)
61    module mki2c (Ifc_i2c_axi4lite#(`paddr, 32, 0));
62      let core_clock<-exposeCurrentClock;
63      let core_reset<-exposeCurrentReset;
64      let ifc();
65      mki2c_axi4lite#(core_clock, core_reset) _temp(ifc);
66      return ifc;
67    endmodule
68
69    (*synthesize*)
```

to

```
Welcome | Soc.bsv | mixed_cluster.bsv | uart_cluster.bsv | Soc.defines | Settings
mixed_cluster.bsv > {} mixed_cluster > Ifc_mixed_cluster > interrupts
31  package mixed_cluster;
41    import plic :: * ;
42    import pinmux :: * ;
43    import pinmux_axi4lite :: * ;
44    `include "Soc.defines"
45
46    interface Ifc_mixed_cluster;
47      method I2C_out i2c0_out;                    //I2c IO interface
48      method I2C_out i2c1_out;                    //I2c IO interface
49      method Bit#(1) sb_ext_interrupt;
50      (*always_ready, always_enabled*)
51      interface GPIO#(32) gpio_io;              //GPIO IO interface
52      interface IOCellSide pinmuxtop_iocell_side;
53      interface PeripheralSide pinmuxtop_peripheral_side;
54      (*always_ready, always_enabled*)
55      method Action interrupts(Bit#(13) inp);
56      interface AXI4_Lite_Slave_IFC#(`paddr, 32, 0) slave;
57      interface AXI4_Lite_Master_IFC#(`paddr, 32, 0) xadc_master;
58    endinterface
59
60    (*synthesize*)
61    module mki2c (Ifc_i2c_axi4lite#(`paddr, 32, 0));
62      let core_clock<-exposeCurrentClock;
63      let core_reset<-exposeCurrentReset;
64      let ifc();
65      mki2c_axi4lite#(core_clock, core_reset) _temp(ifc);
66      return ifc;
67    endmodule
68
69    (*synthesize*)
```

Step 2 (interrupts passed as input to PLIC)*:*

Add one more PLIC interrupt i.e. Change the line from

```
Welcome | Soc.bsv | mixed_cluster.bsv | uart_cluster.bsv | Soc.defines | Settings
mixed_cluster.bsv > ...
uart0 Aa ab .* No results
112
113     AXI4_Lite_Master_Xactor_IFC #(`paddr, 32, 0) c2m_xactor <- mkAXI4_Lite_Master_Xactor;
114     AXI4_Lite_Slave_Xactor_IFC #(`paddr, 32, 0) c2s_xactor <- mkAXI4_Lite_Slave_Xactor;
115     AXI4_Lite_Fabric_IFC #(`MixedCluster_Num_Masters, `MixedCluster_Num_Slaves, `paddr, 32,0)
116                                                                  fabric <- mkAXI4_Lite_Fabric(fn_slave_map);
117     let i2c0 <- mki2c;
118     let i2c1 <- mki2c;
119     let gpio <- mkgpio();
120     let plic <- mkplic();
121     let pinmuxtop <- mkpinmuxtop();
122     Ifc_err_slave_axi4lite#(`paddr, 32, 0 ) err_slave <- mkerr_slave_axi4lite;
123     Wire#(Bit#(12)) wr_external_interrupts <- mkDWire('d0);
124     Wire#(Bit#(1)) wr_sb_ext_interrupt <- mkDWire(0);
125
126     //Rule to connect PLIC interrupt to the core's sideband
127     rule rl_core_plic_connection;
128       let {lv_plic_intr, x}<- plic.intrpt_note_sb.get;
129       wr_sb_ext_interrupt <= pack(lv_plic_intr);
130     endrule
131
132     rule rl_connect_plic_connections;  get
133       let tmp <- gpio.sb_gpio_to_plic.get;
134       Bit#(16) lv_gpio_intr= truncate(pack(tmp));
135       Bit#(31) plic_inputs= {wr_external_interrupts[11:6], i2c1.isint, i2c0.isint, lv_gpio_intr, wr_external_interrupts[5:0],0};
136       plic.ifc_external_irq_io(plic_inputs);
137     endrule
138
139     mkConnection(c2m_xactor.axi_side, fabric.v_from_masters[0]);
140
141     mkConnection(c2s_xactor.o_wr_addr,c2m_xactor.i_wr_addr);
142     mkConnection(c2s_xactor.o_wr_data,c2m_xactor.i_wr_data);
```

to

```
AXI4_Lite_Master_Xactor_IFC #(`paddr, 32, 0) c2m_xactor <- mkAXI4_Lite_Master_Xactor;
AXI4_Lite_Slave_Xactor_IFC #(`paddr, 32, 0) c2s_xactor <- mkAXI4_Lite_Slave_Xactor;
AXI4_Lite_Fabric_IFC #(`MixedCluster_Num_Masters, `MixedCluster_Num_Slaves, `paddr, 32,0)
                                                                                fabric <- mkAXI4_Lite_Fabric(fn_slave_map);
let i2c0 <- mki2c;
let i2c1 <- mki2c;
let gpio <- mkgpio();
let plic <- mkplic();
let pinmuxtop <- mkpinmuxtop();
Ifc_err_slave_axi4lite#(`paddr, 32, 0 ) err_slave <- mkerr_slave_axi4lite;
Wire#(Bit#(12)) wr_external_interrupts <- mkDWire('d0);
Wire#(Bit#(1)) wr_sb_ext_interrupt <- mkDWire(0);

//Rule to connect PLIC interrupt to the core's sideband
rule rl_core_plic_connection;
  let {lv_plic_intr, x}<- plic.intrpt_note_sb.get;
  wr_sb_ext_interrupt <= pack(lv_plic_intr);
endrule

rule rl_connect_plic_connections;
  let tmp <- gpio.sb_gpio_to_plic.get;
  Bit#(16) lv_gpio_intr= truncate(pack(tmp));
  Bit#(32) plic_inputs= {wr_external_interrupts[12:6], i2c1.isint, i2c0.isint, lv_gpio_intr, wr_external_interrupts[5:0],0};
  plic.ifc_external_irq_io(plic_inputs);
endrule

mkConnection(c2m_xactor.axi_side, fabric.v_from_masters[0]);

mkConnection(c2s_xactor.o_wr_addr,c2m_xactor.i_wr_addr);
mkConnection(c2s_xactor.o_wr_data,c2m_xactor.i_wr_data);
```

Step 3: Add one more interrupt to PLIC module

Change the number of arguments passed to PLIC module by changing the code from

Welcome | Soc.bsv | mixed_cluster.bsv | plic.bsv | uart_cluster.bsv | Soc.defines | Settings

mixed_cluster.bsv > {} mixed_cluster > {} mkplic

```
31  package mixed_cluster;
60    (*synthesize*)
62      let core_clock<-exposeCurrentClock;
63      let core_reset<-exposeCurrentReset;
64      let ifc();
65      mki2c_axi4lite#(core_clock, core_reset) _temp(ifc);
66      return ifc;
67    endmodule
68
69    (*synthesize*)
70    module mkgpio(Ifc_gpio_axi4lite#(`paddr, 32, 0, 32));
71      let ifc();
72      mkgpio_axi4lite _temp(ifc);
73      return ifc;
74    endmodule
75
76    (*synthesize*)
77    module mkplic(Ifc_plic_axi4lite#(`paddr, 32, 0, 31, 2, 0));
78      let ifc();
79      mkplic_axi4lite#(`PLICBase)_temp(ifc);
80      return ifc;
81    endmodule
82
83    (*synthesize*)
84    module mkpinmuxtop(Ifc_pinmux_axi4lite#(`paddr, 32, 0));
85      let ifc();
86      mkpinmux_axi4lite _temp(ifc);
87      return ifc;
88    endmodule
```

to

Welcome | Soc.bsv | mixed_cluster.bsv | plic.bsv | uart_cluster.bsv | Soc.defines | Settings

mixed_cluster.bsv > {} mixed_cluster > {} mkplic

```
31  package mixed_cluster;
60    (*synthesize*)
62      let core_clock<-exposeCurrentClock;
63      let core_reset<-exposeCurrentReset;
64      let ifc();
65      mki2c_axi4lite#(core_clock, core_reset) _temp(ifc);
66      return ifc;
67    endmodule
68
69    (*synthesize*)
70    module mkgpio(Ifc_gpio_axi4lite#(`paddr, 32, 0, 32));
71      let ifc();
72      mkgpio_axi4lite _temp(ifc);
73      return ifc;
74    endmodule
75
76    (*synthesize*)
77    module mkplic(Ifc_plic_axi4lite#(`paddr, 32, 0, 32, 2, 0));
78      let ifc();
79      mkplic_axi4lite#(`PLICBase)_temp(ifc);
80      return ifc;
81    endmodule
82
83    (*synthesize*)
84    module mkpinmuxtop(Ifc_pinmux_axi4lite#(`paddr, 32, 0));
85      let ifc();
86      mkpinmux_axi4lite _temp(ifc);
87      return ifc;
88    endmodule
```

Step 4 (add one more bit to take the interrupt from soc.bsv):

Add one more interrupt bit to “wr_external_interrupts” i.e. change the code from

```
  return slave_num;
endfunction:fn_slave_map

(*synthesize*)
module mkmixed_cluster(Ifc_mixed_cluster);

  AXI4_Lite_Master_Xactor_IFC #(`paddr, 32, 0) c2m_xactor <- mkAXI4_Lite_Master_Xactor;
  AXI4_Lite_Slave_Xactor_IFC #(`paddr, 32, 0) c2s_xactor <- mkAXI4_Lite_Slave_Xactor;
  AXI4_Lite_Fabric_IFC #(`MixedCluster_Num_Masters, `MixedCluster_Num_Slaves, `paddr, 32,0)
                                                                  fabric <- mkAXI4_Lite_Fabric(fn_slave_map);
  let i2c0 <- mki2c;
  let i2c1 <- mki2c;
  let gpio <- mkgpio();
  let plic <- mkplic();
  let pinmuxtop <- mkpinmuxtop();
  Ifc_err_slave_axi4lite#(`paddr, 32, 0 ) err_slave <- mkerr_slave_axi4lite;
  Wire#(Bit#(12)) wr_external_interrupts <- mkDWire('d0);
  Wire#(Bit#(1)) wr_sb_ext_interrupt <- mkDWire(0);
```

to

```
  return slave_num;
endfunction:fn_slave_map

(*synthesize*)
module mkmixed_cluster(Ifc_mixed_cluster);

  AXI4_Lite_Master_Xactor_IFC #(`paddr, 32, 0) c2m_xactor <- mkAXI4_Lite_Master_Xactor;
  AXI4_Lite_Slave_Xactor_IFC #(`paddr, 32, 0) c2s_xactor <- mkAXI4_Lite_Slave_Xactor;
  AXI4_Lite_Fabric_IFC #(`MixedCluster_Num_Masters, `MixedCluster_Num_Slaves, `paddr, 32,0)
                                                                                fabric <- mkAXI4_Lite_Fabric(fn_slave_map);
  let i2c0 <- mki2c;
  let i2c1 <- mki2c;
  let gpio <- mkgpio();
  let plic <- mkplic();
  let pinmuxtop <- mkpinmuxtop();
  Ifc_err_slave_axi4lite#(`paddr, 32, 0 ) err_slave <- mkerr_slave_axi4lite;
  Wire#(Bit#(13)) wr_external_interrupts <- mkDWire('d0);
  Wire#(Bit#(1)) wr_sb_ext_interrupt <- mkDWire(0);
```

Step 5: Update the method "inp" to take one more interrupt.

Change the code from

```
  mkConnection (fabric.v_to_slaves [`I2C0_slave_num ],    i2c0.slave);
  mkConnection (fabric.v_to_slaves [`I2C1_slave_num ],    i2c1.slave);
  mkConnection (fabric.v_to_slaves [`PLIC_slave_num ], plic.slave);
  mkConnection (fabric.v_to_slaves [`GPIO_slave_num ], gpio.slave);
  mkConnection (fabric.v_to_slaves [`Pinmux_slave_num ], pinmuxtop.slave);
  mkConnection (fabric.v_to_slaves [`MixedCluster_err_slave_num ] , err_slave.slave);

  method I2C_out i2c0_out= i2c0.io;
  method I2C_out i2c1_out= i2c1.io;
  method sb_ext_interrupt = wr_sb_ext_interrupt;
  interface gpio_io= gpio.io;
  method Action interrupts(Bit#(12) inp);
    wr_external_interrupts<= inp;
  endmethod
  interface pinmuxtop_iocell_side = pinmuxtop.pinmuxaxi4lite_iocell_side;
```

to

```
  mkConnection (fabric.v_to_slaves [`I2C0_slave_num ],    i2c0.slave);
  mkConnection (fabric.v_to_slaves [`I2C1_slave_num ],    i2c1.slave);
  mkConnection (fabric.v_to_slaves [`PLIC_slave_num ], plic.slave);
  mkConnection (fabric.v_to_slaves [`GPIO_slave_num ], gpio.slave);
  mkConnection (fabric.v_to_slaves [`Pinmux_slave_num ], pinmuxtop.slave);
  mkConnection (fabric.v_to_slaves [`MixedCluster_err_slave_num ] , err_slave.slave);

  method I2C_out i2c0_out= i2c0.io;
  method I2C_out i2c1_out= i2c1.io;
  method sb_ext_interrupt = wr_sb_ext_interrupt;
  interface gpio_io= gpio.io;
  method Action interrupts(Bit#(13) inp);
    wr_external_interrupts<= inp;
  endmethod
  interface pinmuxtop_iocell_side = pinmuxtop.pinmuxaxi4lite_iocell_side;
```

5. Make the following changes in the “fpga_top.v”.

Step 1 (add io pad declarations for the UART IO pins):

Add the IO pins for the new UART (depends on the IP being ported). Add two IO pads for UART3 (similar to UART0 IO pads) i.e. change the following code from

```
  input phy_col,
  input phy_rx_er,
  output phy_rst_n,
  output phy_tx_en,
  output [3:0]phy_tx_data,
  output phy_ref_clk,
//  input phy_mdio_i,
//  output phy_mdio_o,
  inout phy_mdio,
//  output phy_mdio_t,
  output phy_mdc,
  // ---- JTAG ports ------- //
  // ---- UART ports --------//
    input         uart0_SIN,
    output        uart0_SOUT,

//   // ---- I2C ports --------//
    inout         i2c0_sda,
    inout         i2c0_scl,
    inout         i2c1_sda,
    inout         i2c1_scl,
```

to

```
  input phy_col,
  input phy_rx_er,
  output phy_rst_n,
  output phy_tx_en,
  output [3:0]phy_tx_data,
  output phy_ref_clk,
//  input phy_mdio_i,
//  output phy_mdio_o,
  inout phy_mdio,
//  output phy_mdio_t,
  output phy_mdc,
  // ---- JTAG ports ------- //
  // ---- UART ports --------//
    input        uart0_SIN,
    output       uart0_SOUT,
    input        uart3_SIN,
    output       uart3_SOUT,

//   // ---- I2C ports --------//
    inout        i2c0_sda,
    inout        i2c0_scl,
```

Step 2:
Add new UART IO pins in mkSoc core instance:
Case 1: the IO pins are uni directional (either input or output):
Map mkSoc pins with fpga_top io pins i.e. change the code from

```
    // SPI ports
      .spi0_io_mosi(spi0_mosi),
      .spi0_io_sclk(spi0_sclk),
      .spi0_io_nss(spi0_nss),
      .spi0_io_miso_dat(spi0_miso),
      /*.spi1_io_mosi(spi1_mosi),
      .spi1_io_sclk(spi1_sclk),
      .spi1_io_nss(spi1_nss),
      .spi1_io_miso_dat(spi1_miso),*/

     // UART port definitions
        .uart0_io_SIN(uart0_SIN),
        .uart0_io_SOUT(uart0_SOUT),
     // AXI4 Master interface to be connected to DDR3
```

to

```
// SPI ports
  .spi0_io_mosi(spi0_mosi),
  .spi0_io_sclk(spi0_sclk),
  .spi0_io_nss(spi0_nss),
  .spi0_io_miso_dat(spi0_miso),
  /*.spi1_io_mosi(spi1_mosi),
  .spi1_io_sclk(spi1_sclk),
  .spi1_io_nss(spi1_nss),
  .spi1_io_miso_dat(spi1_miso),*/

 // UART port definitions
 ........uart0_io_SIN(uart0_SIN),
 ........uart0_io_SOUT(uart0_SOUT),
 ........uart3_io_SIN(uart3_SIN),
 ........uart3_io_SOUT(uart3_SOUT),
 // AXI4 Master interface to be connected to DDR3
```

Case 2: The IO pin is bidirectional
This case will be covered in the section “adding ethernet lite IP” section.

6. Make the following changes in the “constraints.xdc”

The signal to pin mapping is done in the constraints.xdc file.
Step 1 (add signal to pin mapping for UART IO pins):
Add the pin mapping for the UART 3 IO pins similar to UART 0 IO pins i.e. change the following code from

```
## Pmod Header JC
#set_property -dict { PACKAGE_PIN U12   IOSTANDARD LVCMOS33 } [get_ports { spi1_nss }]; #IO_L20P_T3_A08_D24_14 Sch=jc_p[1]
#set_property -dict { PACKAGE_PIN V12   IOSTANDARD LVCMOS33 } [get_ports { spi1_mosi }]; #IO_L20N_T3_A07_D23_14 Sch=jc_n[1]
#set_property -dict { PACKAGE_PIN V10   IOSTANDARD LVCMOS33 } [get_ports { spi1_miso }]; #IO_L21P_T3_DQS_14 Sch=jc_p[2]
#set_property -dict { PACKAGE_PIN U14   IOSTANDARD LVCMOS33 } [get_ports { spi1_sclk ] }]; #IO_L22P_T3_A05_D21_14 Sch=jc_p[3]

## USB-UART Interface
set_property -dict { PACKAGE_PIN D10   IOSTANDARD LVCMOS33 } [get_ports { uart0_SOUT }]; #IO_L19N_T3_VREF_16 Sch=uart_rxd_out
set_property -dict { PACKAGE_PIN A9    IOSTANDARD LVCMOS33 } [get_ports { uart0_SIN }]; #IO_L14N_T2_SRCC_16 Sch=uart_txd_in

## ChipKit Outer Digital Header
set_property -dict { PACKAGE_PIN V15   IOSTANDARD LVCMOS33 } [get_ports { io7_cell  }]; #IO_L16P_T2_CSI_B_14 Sch=ck_io[0]
set_property PULLDOWN true [get_ports { io7_cell }];
```

to

```
## Pmod Header JC
#set_property -dict { PACKAGE_PIN U12   IOSTANDARD LVCMOS33 } [get_ports { spi1_nss }]; #IO_L20P_T3_A08_D24_14 Sch=jc_p[1]
#set_property -dict { PACKAGE_PIN V12   IOSTANDARD LVCMOS33 } [get_ports { spi1_mosi }]; #IO_L20N_T3_A07_D23_14 Sch=jc_n[1]
#set_property -dict { PACKAGE_PIN V10   IOSTANDARD LVCMOS33 } [get_ports { spi1_miso }]; #IO_L21P_T3_DQS_14 Sch=jc_p[2]
#set_property -dict { PACKAGE_PIN U14   IOSTANDARD LVCMOS33 } [get_ports { spi1_sclk ] }]; #IO_L22P_T3_A05_D21_14 Sch=jc_p[3]

## USB-UART Interface
set_property -dict { PACKAGE_PIN D10   IOSTANDARD LVCMOS33 } [get_ports { uart0_SOUT }]; #IO_L19N_T3_VREF_16 Sch=uart_rxd_out
set_property -dict { PACKAGE_PIN A9    IOSTANDARD LVCMOS33 } [get_ports { uart0_SIN }]; #IO_L14N_T2_SRCC_16 Sch=uart_txd_in
set_property -dict { PACKAGE_PIN xxx   IOSTANDARD LVCMOS33 } [get_ports { uart3_SOUT }]; #IO_L19N_T3_VREF_16 Sch=uart_rxd_out
set_property -dict { PACKAGE_PIN yyy   IOSTANDARD LVCMOS33 } [get_ports { uart3_SIN }]; #IO_L14N_T2_SRCC_16 Sch=uart_txd_in

## ChipKit Outer Digital Header
```

Note *: In the above code the pins “**xxx**” & “**yyy**” are depending on the Designers choice with the selected FPGA part number.

This completes the adding IP in one of the existing cluster in SP2020 to the core.

Integrating Fast peripherals with Shakti:

Next we will be looking into adding the high speed AXI4 link directly. We assume we are having a module called AES and trying to add the same to the core.

AES tends to operate in a high frequency mostly in equivalence with the core frequency. So AES will be integrated with the SoC using AXI interface (fast fabric).

The AES is integrated through AXI4 fast fabric, so the memory allocation for AES doesn't follow any hierarchy like it is there in the UART. The memory mapping for AES should be unique in that it doesn't fall in any of the range that is already in usage.

First clone **crypto-box** repository for AES,

$ cd sp2020/c64-a100/
$ git clone https://gitlab.com/shaktiproject/cores/crypto-box

Add path for the AES in **bsvpath** file,
crypto-box/aes_buffer

Add the following command in **Makefile**

```
    @cp ${BS_VERILOG_LIB}/ResetEither.v ${VERILOGDIR}
    @cp ${BS_VERILOG_LIB}/MakeReset0.v ${VERILOGDIR}
    @cp ${BS_VERILOG_LIB}/SyncReset0.v ${VERILOGDIR}
    @cp ${BS_VERILOG_LIB}/ClockInverter.v ${VERILOGDIR}
    @cp ${BS_VERILOG_LIB}/SyncFIFO1.v ${VERILOGDIR}
    @cp ${BS_VERILOG_LIB}/RevertReg.v ${VERILOGDIR}
    @cp ${BS_VERILOG_LIB}/BRAM1.v ${VERILOGDIR}
    @cp ${BS_VERILOG_LIB}/BRAM2.v ${VERILOGDIR}
    @cp ./common_verilog/bram_1rw.v ${VERILOGDIR}
    @cp ${BS_VERILOG_LIB}/FIFO20.v ${VERILOGDIR}
    @cp ./common_verilog/bram_2rw.v ${VERILOGDIR}
    @cp common_verilog/signedmul.v ${VERILOGDIR}
    @cp ${BS_VERILOG_LIB}/SyncRegister.v ${VERILOGDIR}
```

Hierarchy

For AES the hierarchy is as follows (leftmost is the top file),

Soc.bsv is the top file in the case of AES

```
// peripheral imports
import clint::*;
import err_slave::*;
import pwm_cluster :: * ;
import uart_cluster :: * ;
import spi_cluster :: * ;
import mixed_cluster :: * ;
import buffer_aes_read_write_req::*; //AES
import uart :: *;
import spi :: *;
import pwm :: *;
import i2c :: *;
import gpio :: *;
import bram :: *;
import debug_types::*;
import xilinxdtm::*;
import riscvDebug013::*;
import debug_halt_loop::*;
import debug_types::*;
import pinmux::*;
```

All the Memory mapping details will be in the **Soc.defines** file.

Increase the **Num_Fast_Slaves** to **6** as below,

```
`define Num_Fast_Slaves 6
`define Memory_slave_num 0
`define Clint_slave_num 1
`define Debug_slave_num 2
`define FastErr_slave_num 3
`define Slow_fabric_slave_num 4
`define AESBUF_slave_num 5
```

```
`define I2C0Base     'h0004_0000
`define I2C0End      'h0004_00FF
`define GPIOBase       'h0004_0100
`define GPIOEnd        'h0004_01FF
`define XADCBase     'h0004_1000
`define XADCEnd      'h0004_13FF
`define I2C1Base     'h0004_1400
`define I2C1End      'h0004_14FF
`define PinmuxBase       'h0004_1500
`define PinmuxConfigReg 'h0004_1510
`define PinmuxEnd        'h0004_15FF
`define PLICBase      'h0C00_0000
`define PLICEnd  'h0C01_001F
`define AESBUFBase 'h0006_1400
`define AESBUFEnd  'h0006_14FF
```

In Soc file add the memory mapping of the AES instance to the function which will return the instance (slave number),

```
function Bit#(TLog#(`Num_Fast_Slaves)) fn_slave_map_fast (Bit#(`paddr) addr);
  Bit#(TLog#(`Num_Fast_Slaves)) slave_num = 0;
  if(addr >= `MemoryBase && addr<= `MemoryEnd)
    slave_num = `Memory_slave_num;
  else if(addr >= `ClintBase && addr <= `ClintEnd)
    slave_num = `Clint_slave_num;
  else if(addr >= `DebugBase && addr <= `DebugEnd)
    slave_num = `Debug_slave_num;
  else if(addr >= `SlowBase && addr <= `SlowEnd)
    slave_num = `Slow_fabric_slave_num;
  else if(addr >= `PLICBase && addr <= `PLICEnd)
    slave_num = `Slow_fabric_slave_num;
  else if(addr>=`AESBUFBase && addr<= `AESBUFEnd)
    slave_num= `AESBUF_slave_num;
  else
    slave_num = `FastErr_slave_num;

  return slave_num;
endfunction:fn_slave_map_fast
```

Declare the AES module and pass the required parameters like address width, data width, user width, clock and reset,

Instantiate the interface of AES inside the module definition of Soc,

```
Ifc_pwm_cluster pwm_cluster <- mkpwm_cluster;
Ifc_uart_cluster uart_cluster <- mkuart_cluster;
Ifc_spi_cluster spi_cluster <- mkspi_cluster;
Ifc_mixed_cluster mixed_cluster <- mkmixed_cluster;
Ifc_aesbuf_axi4#(`paddr, 64, 0) aesbuf <-mkaesbuf_axi4(curr_clk, curr_reset);
Ifc_err_slave_axi4lite#(`paddr,32,0) err_slave <- mkerr_slave_axi4lite;
Ifc_bram_axi4lite#(`paddr, 32, 0,  13) boot <- mkbram_axi4lite('h1000, "boot.mem","Boot");
Wire#(Bit#(3)) wr_ext_interrupts <- mkWire();
```

The curr_clk and curr_reset is the clock and reset which the core gets, the AES operates at the frequency at which the core operates in this case. Connect the AXI4 interface of AES to AXI4 fast fabric,

```
mkConnection (fabric.v_to_slaves [`Clint_slave_num ],clint.slave);
mkConnection (fabric.v_to_slaves [`FastErr_slave_num ] , fast_err_slave.slave);
mkConnection (fabric.v_to_slaves [`Debug_slave_num ] , debug_memory.slave);
mkConnection (fabric.v_to_slaves [`Slow_fabric_slave_num], slow_fabric.v_from_masters[0]);
mkConnection (fabric.v_to_slaves [`AESBUF_slave_num], aesbuf.slave);
```

This completes the integrating fast IP in SP2020 to the core.

Integration of Ethernet Lite - AXI4 Lite Slave

AXI Ethernet Lite is an IP given by Xilinx which is integrated at fpga_top level. It acts as an AXI4 LITE slave and is integrated using AXI4 Lite which is a slow fabric. The slave must be added to fabric in Soc.bsv file and the memory mapping for the slave, adding of a slave number must be given in Soc.defines file.

In Soc.bsv file, add slave_num for the new slave

```
function Bit#(TLog#(`Num_Slaves)) fn_slave_map (Bit#(`paddr) addr);
  Bit#(TLog#(`Num_Slaves)) slave_num = 0;
  if(addr >= `PWMClusterBase && addr <= `PWMClusterEnd)
    slave_num = `PWMCluster_slave_num;
  else if(addr >= `UARTClusterBase && addr <= `UARTClusterEnd)
    slave_num = `UARTCluster_slave_num;
  else if(addr >= `SPIClusterBase && addr <= `SPIClusterEnd)
    slave_num = `SPICluster_slave_num;
  else if(addr >= `MixedClusterBase && addr <= `MixedClusterEnd)
    slave_num = `MixedCluster_slave_num;
  else if(addr >= `PLICBase && addr <= `PLICEnd)
    slave_num = `MixedCluster_slave_num;
  else if(addr >= `BootBase && addr <= `BootEnd)
    slave_num = `Boot_slave_num;
  else if (addr >= `EthBase && addr <= `EthEnd)
    slave_num = `Eth_slave_num;
  else
    slave_num = `Err_slave_num;

  return slave_num;
endfunction:fn_slave_map
```

Declare eth_master interface:

```
  interface Ifc_Soc;
    interface Ifc_spi_io spi0_io;
//    interface Ifc_spi_io spi2_io;
    interface RS232 uart0_io;
    method I2C_out i2c0_out;                    //I2c IO interface
    method I2C_out i2c1_out;                    //I2c IO interface
    (*always_ready, always_enabled*)  paddr
    interface AXI4_Lite_Master_IFC#(`paddr, 32, 0) xadc_master;
    interface AXI4_Lite_Master_IFC#(`paddr, 32, 0) eth_master;
    interface AXI4_Master_IFC#(`paddr, ELEN, 0) mem_master;
    interface IOCellSide iocell_io;
```

Connect eth_master interface to slow fabric.

```
    interface iocell_io = mixed_cluster.pinmuxtop_iocell_side;
    interface xadc_master = mixed_cluster.xadc_master;
    interface eth_master = slow_fabric.v_to_slaves[`Eth_slave_num];
    interface mem_master = fabric.v_to_slaves [`Memory_slave_num];
    method Action ext_interrupts(Bit#(3) i);
      wr_ext_interrupts <= i;
    endmethod
  `ifdef rtldump
    interface io_dump= eclass.io_dump;
  `endif

  endmodule: mkSoc
endpackage: Soc
```

In Soc.defines file,

```
`define Num_Slaves 7
`define PWMCluster_slave_num 0
`define UARTCluster_slave_num 1
`define SPICluster_slave_num 2
`define MixedCluster_slave_num 3
`define Boot_slave_num 4
`define Eth_slave_num 5
`define Err_slave_num 6
```

1. Increment value of Num_Slaves by 1.
2. Add `define Eth_slave_num *n* (replace *n* by the last slave number -1)
3. Change `define Err_slave_num *x* to `define Err_slave_num *x+1* (*x* being the original value)

```
`define BootBase          'h0000_1000
`define BootEnd           'h0000_2FFF
`define UARTClusterBase   'h0001_1300
`define UARTClusterEnd    'h0001_1540
`define SPIClusterBase    'h0002_0000
`define SPIClusterEnd     'h0002_01FF
`define PWMClusterBase    'h0003_0000
`define PWMClusterEnd     'h0003_05FF
`define MixedClusterBase  'h0004_0000
`define MixedClusterEnd   'h0004_15FF
`define EthBase           'h0004_4000
`define EthEnd            'h0004_7FFF
```

4. Add `define EthBase ‘addr1 and `define EthEnd ‘addr2 . Define addr1 and addr2 such that it does not conflict with other slave addresses and also is within range of SlowBase and SlowEnd

- make generate_verilog : to generate updated verilog files
- In fpga_top, the newly generated core's verilog file will now have the ports corresponding to eth_master slave.

Generated mkSoc.v with newly added ports for ethernet slave:

```
311     eth_master_awvalid,

312

313     eth_master_awaddr,

314

315     eth_master_awprot,

316

317     eth_master_awsize,

318

319     eth_master_m_awready_awready,

320

321     eth_master_wvalid,

322

323     eth_master_wdata,

324

325     eth_master_wstrb,

326

327     eth_master_m_wready_wready,

328

329     eth_master_m_bvalid_bvalid,
330     eth_master_m_bvalid_bresp,

331

332     eth_master_bready,

333

334     eth_master_arvalid,

335

336     eth_master_araddr,

337

338     eth_master_arprot,

339

340     eth_master_arsize,

341

342     eth_master_m_arready_arready,

343

344     eth_master_m_rvalid_rvalid,
345     eth_master_m_rvalid_rresp,
346     eth_master_m_rvalid_rdata,

347

348     eth_master_rready,

349

350         mem_master_AWVALID,
```

- Open project (.xpr file) in Vivado GUI
- Connect the signals from the Ethernet Lite IP to these newly generated ports ports using wires to integrate the third party IP to the SoC

Integrating in fpga_top.v:

Initialise new ports:

```
  // ethernet ports
  input phy_tx_clk,
  input phy_rx_clk,
  input phy_crs,
  input phy_dv,
  input [3:0]phy_rx_data,
  input phy_col,
  input phy_rx_er,
  output phy_rst_n,
  output phy_tx_en,
  output [3:0]phy_tx_data,
  output phy_ref_clk,
//  input phy_mdio_i,
//  output phy_mdio_o,
  inout phy_mdio,
//  output phy_mdio_t,
  output phy_mdc,
```

Initialise all wires:

```
    wire [1 : 0] xadc_master_m_rvalid_rresp;
    wire [32-1 : 0] xadc_master_m_rvalid_rdata;
    wire xadc_master_rready;
    // ETH Axi4-Lite Slave
    wire eth_master_awvalid;
    wire [13-1 : 0] eth_master_awaddr;
    wire eth_master_m_awready_awready;
    wire eth_master_wvalid;
    wire [32-1 : 0] eth_master_wdata;
    wire [(32/8)-1 : 0] eth_master_wstrb;
    wire eth_master_m_wready_wready;
    wire eth_master_m_bvalid_bvalid;
    wire [1:0] eth_master_m_bvalid_bresp;
    wire eth_master_bready;
    wire eth_master_arvalid;
    wire [13-1 : 0] eth_master_araddr;
    wire eth_master_m_arready_arready;
    wire eth_master_m_rvalid_rvalid;
    wire [1 : 0] eth_master_m_rvalid_rresp;
    wire [32-1 : 0] eth_master_m_rvalid_rdata;
    wire eth_master_rready;
  // --------- Address width truncation and Reset gene
  wire [31:0] temp_s_axi_awaddr, temp_s_axi_araddr;
```

Within

mkSoc core(
);

```
545    // ----- Instantiating the C-class SoC -------------//
546  mkSoc core(
547        // Main Clock and Reset to the SoC
```

Add the ports for new slave.

```
        .iocell_io_io20_cell_outen(io20_cell_en),

        // ETH connection
        .eth_master_awvalid(eth_master_awvalid),
        .eth_master_awaddr(eth_master_awaddr),
        .eth_master_m_awready_awready(eth_master_m_awready_awready),
        .eth_master_wvalid(eth_master_wvalid),
        .eth_master_wdata(eth_master_wdata),
        .eth_master_wstrb(eth_master_wstrb),
        .eth_master_m_wready_wready(eth_master_m_wready_wready),
        .eth_master_m_bvalid_bvalid(eth_master_m_bvalid_bvalid),
        .eth_master_m_bvalid_bresp(eth_master_m_bvalid_bresp),
        .eth_master_bready(eth_master_bready),
        .eth_master_arvalid(eth_master_arvalid),
        .eth_master_araddr(eth_master_araddr),
        .eth_master_m_arready_arready(eth_master_m_arready_arready),
        .eth_master_m_rvalid_rvalid(eth_master_m_rvalid_rvalid),
        .eth_master_m_rvalid_rresp(eth_master_m_rvalid_rresp),
        .eth_master_m_rvalid_rdata(eth_master_m_rvalid_rdata),
        .eth_master_rready(eth_master_rready),

        // XADC connection
        .xadc_master_awvalid(xadc_master_awvalid),
        .xadc_master_awaddr(xadc_master_awaddr),
```

Interfacing with AXI-Ethernet Lite IP:

After adding and configuring the AXI-Ethernet Lite IP from IP Catalog (in Vivado GUI), add the IP instantiation in the fpga_top.v file.

```
768 axi_ethernetlite_0 eth10 (
769   .s_axi_aclk(core_clk),          // input wire s_axi_aclk
770   .s_axi_aresetn(~soc_reset),   // input wire s_axi_aresetn
771   .ip2intc_irpt(eth10_ip2intc_irpt),     // output wire ip2intc_irpt
772   .s_axi_awaddr     (eth_master_awaddr),
773   .s_axi_awvalid    (eth_master_awvalid),
774   .s_axi_awready    (eth_master_m_awready_awready),
775   .s_axi_wdata      (eth_master_wdata),
776   .s_axi_wstrb      (eth_master_wstrb),
777   .s_axi_wvalid     (eth_master_wvalid),
778   .s_axi_wready     (eth_master_m_wready_wready),
779   .s_axi_bresp      (eth_master_m_bvalid_bresp),
780   .s_axi_bvalid     (eth_master_m_bvalid_bvalid),
781   .s_axi_bready     (eth_master_bready),
782   .s_axi_araddr     (eth_master_araddr),
783   .s_axi_arvalid    (eth_master_arvalid),
784   .s_axi_arready    (eth_master_m_arready_arready),
785   .s_axi_rdata      (eth_master_m_rvalid_rdata),
786   .s_axi_rresp      (eth_master_m_rvalid_rresp),
787   .s_axi_rvalid     (eth_master_m_rvalid_rvalid),
788   .s_axi_rready     (eth_master_rready),
789   .phy_tx_clk(phy_tx_clk),         // input wire phy_tx_clk
790   .phy_rx_clk(phy_rx_clk),         // input wire phy_rx_clk
791   .phy_crs(phy_crs),               // input wire phy_crs
792   .phy_dv(phy_dv),                 // input wire phy_dv
793   .phy_rx_data(phy_rx_data),       // input wire [3 : 0] phy_rx_data
794   .phy_col(phy_col),               // input wire phy_col
795   .phy_rx_er(phy_rx_er),           // input wire phy_rx_er
796   .phy_rst_n(phy_rst_n),           // output wire phy_rst_n
797   .phy_tx_en(phy_tx_en),           // output wire phy_tx_en
798   .phy_tx_data(phy_tx_data),       // output wire [3 : 0] phy_tx_data
799   .phy_mdio_i(phy_mdio_i),         // input wire phy_mdio_i
800   .phy_mdio_o(phy_mdio_o),         // output wire phy_mdio_o
801   .phy_mdio_t(phy_mdio_t),         // output wire phy_mdio_t
802   .phy_mdc(phy_mdc)                // output wire phy_mdc
803 );
```

In Constraints, add the required ports with the appropriate mapping:

```
123 # SMSC Ethernet PHY
124 set_property -dict { PACKAGE_PIN D17   IOSTANDARD LVCMOS33 } [get_ports { phy_col }]; #IO_L16N_T2_A27_15 Sch=eth_col
125 set_property -dict { PACKAGE_PIN G14   IOSTANDARD LVCMOS33 } [get_ports { phy_crs }]; #IO_L15N_T2_DQS_ADV_B_15 Sch=eth_crs
126 set_property -dict { PACKAGE_PIN F16   IOSTANDARD LVCMOS33 } [get_ports { phy_mdc }]; #IO_L14N_T2_SRCC_15 Sch=eth_mdc
127 set_property -dict { PACKAGE_PIN K13   IOSTANDARD LVCMOS33 } [get_ports { phy_mdio }]; #IO_L17P_T2_A26_15 Sch=eth_mdio
128 set_property -dict { PACKAGE_PIN G18   IOSTANDARD LVCMOS33 } [get_ports { phy_ref_clk }]; #IO_L22P_T3_A17_15
    Sch=eth_ref_clk
129 set_property -dict { PACKAGE_PIN C16   IOSTANDARD LVCMOS33 } [get_ports { phy_rst_n }]; #IO_L20P_T3_A20_15 Sch=eth_rstn
130 set_property -dict { PACKAGE_PIN F15   IOSTANDARD LVCMOS33 } [get_ports { phy_rx_clk }]; #IO_L14P_T2_SRCC_15
    Sch=eth_rx_clk
131 set_property -dict { PACKAGE_PIN G16   IOSTANDARD LVCMOS33 } [get_ports { phy_dv }]; #IO_L13N_T2_MRCC_15 Sch=eth_rx_dv
132 set_property -dict { PACKAGE_PIN D18   IOSTANDARD LVCMOS33 } [get_ports { phy_rx_data[0] }]; #IO_L21N_T3_DQS_A18_15
    Sch=eth_rxd[0]
133 set_property -dict { PACKAGE_PIN E17   IOSTANDARD LVCMOS33 } [get_ports { phy_rx_data[1] }]; #IO_L16P_T2_A28_15
    Sch=eth_rxd[1]
134 set_property -dict { PACKAGE_PIN E18   IOSTANDARD LVCMOS33 } [get_ports { phy_rx_data[2] }]; #IO_L21P_T3_DQS_15
    Sch=eth_rxd[2]
135 set_property -dict { PACKAGE_PIN G17   IOSTANDARD LVCMOS33 } [get_ports { phy_rx_data[3] }]; #IO_L18N_T2_A23_15
    Sch=eth_rxd[3]
136 set_property -dict { PACKAGE_PIN C17   IOSTANDARD LVCMOS33 } [get_ports { phy_rx_er }]; #IO_L20N_T3_A19_15 Sch=eth_rxerr
137 set_property -dict { PACKAGE_PIN H16   IOSTANDARD LVCMOS33 } [get_ports { phy_tx_clk }]; #IO_L13P_T2_MRCC_15
    Sch=eth_tx_clk
138 set_property -dict { PACKAGE_PIN H15   IOSTANDARD LVCMOS33 } [get_ports { phy_tx_en }]; #IO_L19N_T3_A21_VREF_15
    Sch=eth_tx_en
139 set_property -dict { PACKAGE_PIN H14   IOSTANDARD LVCMOS33 } [get_ports { phy_tx_data[0] }]; #IO_L15P_T2_DQS_15
    Sch=eth_txd[0]
140 set_property -dict { PACKAGE_PIN J14   IOSTANDARD LVCMOS33 } [get_ports { phy_tx_data[1] }]; #IO_L19P_T3_A22_15
    Sch=eth_txd[1]
141 set_property -dict { PACKAGE_PIN J13   IOSTANDARD LVCMOS33 } [get_ports { phy_tx_data[2] }]; #IO_L17N_T2_A25_15
    Sch=eth_txd[2]
142 set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 } [get_ports { phy_tx_data[3] }]; #IO_L18P_T2_A24_15
    Sch=eth_txd[3]
143
```

Note:
In sp2020 : c64-a100, the commands for integrating the ethernet lite module have already been included. Kindly look into that for reference.