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.
 - \circ $\;$ 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),

 $Soc \gets uart_cluster$

🗙 Welcome	E Soc.bsv x ≭ Settings
≣ Soc.bsv >	() Soc > () mkSoc
31	package Soc; > uart_cluster Aa ab * 2of11 $\uparrow \downarrow \equiv x$
274	(*synthesize*)
277	<pre>let curr_reset<-exposeCurrentReset;</pre>
278	<pre>Ifc_ccore_axi4 ccore <- mkccore_axi4(`resetpc, 0);</pre>
279	
280	AXI4_Fabric_IFC #(`Num_Fast_Masters, `Num_Fast_Slaves, `paddr, ELEN, USERSPACE)
281	<pre>fabric <- mkAXI4_Fabric (fn_slave_map_fast);</pre>
282	<pre>Ifc_clint_axi4#(`paddr, ELEN, 0, 1, 256) clint <- mkclint_axi4();</pre>
283	<pre>Ifc_debug_halt_loop_axi4#(`paddr, ELEN, USERSPACE) debug_memory <-</pre>
	mkdebug_halt_loop_axi4;
284	<pre>Ifc_err_slave_axi4#(`paddr,ELEN,0) fast_err_slave <- mkerr_slave_axi4;</pre>
285	
286	AXI4_Lite_Fabric_IFC #(`Num_Masters, `Num_Slaves, `paddr, 32, USERSPACE)
287	slow_fabric <- mkAXI4_Lite_Fabric // (fn_slave_map);
288	Ifc_pwm_cluster pwm_cluster <- mkpwm_cluster;
289	<pre>Ifc_uart_cluster uart_cluster <mkuart_cluster;< pre=""></mkuart_cluster;<></pre>
290	<pre>Ifc_spi_cluster spi_cluster <- mkspi_cluster;</pre>
291	<pre>Ifc_mixed_cluster mixed_cluster <- mkmixed_cluster;</pre>
292	<pre>Ifc_err_slave_axi4lite#(`paddr,32,0) err_slave <- mkerr_slave_axi4lite;</pre>
293	<pre>Ifc_bram_axi4lite#(`paddr, 32, 0, 13) boot <- mkbram_axi4lite('h1000, "boot.mem",</pre>
	"Boot");
294	<pre>Wire#(Bit#(3)) wr_ext_interrupts <- mkWire();</pre>
295	
296	<pre>Wire#(Bit#(1)) wr_gpio4_in <- mkDWire(0);</pre>
297	<pre>Wire#(Bit#(1)) wr_gpio7_in <- mkDWire(0);</pre>
298	<pre>Wire#(Bit#(1)) wr_gpio8_in <- mkDWire(0);</pre>
299	<pre>Wire#(Bit#(1)) wr_gpio14_in <- mkDWire(0);</pre>
300	<pre>Wire#(Bit#(1)) wr_gpio15_in <- mkDWire(0);</pre>
201	

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

🗙 Welcome	≣ Soc.b	osv	≣ Soc.defines ×											
			rr_slave_n					uart_cluster	Aa	ab *	No resul	ts 1	≓ ×	
	efine	Slow_	_fabric_sla	ve_num 4										
12														
	efine		•	'h8000_										
	efine		2	'h8FFF_	FFFF									
15 `d	efine	Clint	Base	'h0200_	9000									
16 `d	efine	Clint	End	'h020B_	FFFF									
17 `d	efine	Debug	Base	'h0000_	9010									
18 `d	efine	Debug	JEnd	'h0000_	901F									
19 `d	efine	SlowE	lase	'h0000_	1000									
20 `d	efine	SlowE	nd	'h0004_	7FFF									
21														
22 `d	efine	Num_M	lasters 1											
23														
24 `d	efine	Num S	Slaves 7											
25 `d	efine	PWMCl	uster slav	e num 0										
26 `d	efine	UARTO	luster sla	ve num 1										
27 `d	efine	SPICl	uster slav	e num 2										
28 `d	efine	Mixed	Cluster sl	ave num 3										
29 `d	efine	Boot	slave num	4										
30 `d	efine	Eth s	lave num 5											
31 `d	efine	Errs	lave num 6											
32														
33 `d	efine	BootE	lase	'h0000	1000									
34 `d	efine	BootE	nd	'h0000										
35 `d	efine∙	UARTO	lusterBase	_										
			lusterEnd											
			usterBase	'h0002										
			usterEnd	'h0002_										
			usterBase	'h0003										
40 `d			uctorEnd	160003_										

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.

🗙 Welcom	ne ≣ Soc.bsv ≣ Soc.defines × ₹	E Settings			•••
≡ Soc.de	fines				
5			> uart_cluster	Aa_ab* No results	$\wedge = \times$
6	<pre>`define Num_Fast_Slaves 5</pre>				
7	<pre>`define Memory_slave_num</pre>				
8	`define Clint_slave_num 1				
9	<pre>`define Debug_slave_num 2 `define FeetFeetended"</pre>				
10	<pre>`define FastErr_slave_num</pre>				-
11	`define Slow_fabric_slave	_num 4			
12					
13	<pre>`define MemoryBase</pre>	'h8000_0000			
14	<pre>`define MemoryEnd `define ClietDees</pre>	'h8FFF_FFFF			
15	<pre>`define ClintBase</pre>	'h0200_0000			
16	`define ClintEnd	'h020B_FFFF			
17	`define DebugBase	'h0000_0010			
18	`define DebugEnd	'h0000_001F			
19	<pre>`define SlowBase</pre>	'h0000_1000			
20	`define SlowEnd	'h0004_7FFF			
21					
22	`define Num_Masters 1				
23	define Num Clause 7				
24	<pre>`define Num_Slaves 7 `define NumCluster slave</pre>				
25	<pre>`define PWMCluster_slave_</pre>				
26	`define.UARTCluster_slave	_			
27	<pre>`define SPICluster_slave_</pre>				
28	<pre>`define MixedCluster_slav</pre>	e_num 3			
29	<pre>`define Boot_slave_num 4 `define Fth_clave_num F</pre>				
30	<pre>`define Eth_slave_num 5 `define Err clave num 6</pre>				
31	`define Err_slave_num 6				
32	define ReetRace	1000 1000			
33	<pre>`define BootBase `define BootEnd</pre>	'h0000_1000			
34	<pre>`define BootEnd `define UAPTClusterBase</pre>	'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 \rightarrow UART cluster \rightarrow 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

🗙 Welcom	e ≣ Soc.bsv 🛛 🗄 Soc.defines ● 🚅	Settings				ш
≡ Soc.def	ines	-				
33	`define BootBase	'h0000_1000		> uart_cluster	Aa ab * No results 1	$\downarrow = \times$
34	`define BootEnd	'h0000_2FFF	l			
35	`define.UARTClusterBase	•'h0001_1300				
36	`define UARTClusterEnd · · ·	• 'h0001_1540				
37	`define SPIClusterBase	'h0002_0000				
38	`define SPIClusterEnd	'h0002_01FF				
39	`define PWMClusterBase	'h0003_0000				
40	<pre>`define PWMClusterEnd</pre>	'h0003_05FF				_
41	`define MixedClusterBase	'h0004_0000				
42	<pre>`define MixedClusterEnd</pre>	'h0004_15FF				
43	`define EthBase	'h0004_4000				
44	`define EthEnd	'h0004_7FFF				
45						
46	// - PWM cluster					
47	`define PWMCluster_Num_Sl	aves 7				
48	<pre>`define PWMCluster_Num_Ma</pre>	sters 1				
49						
50	<pre>`define PWM0_slave_num 0</pre>					
51	<pre>`define PWM1_slave_num 1</pre>					
52	<pre>`define PWM2_slave_num 2</pre>					
53	<pre>`define PWM3_slave_num 3</pre>					
54	`define PWM4_slave_num 4					
55	<pre>`define PWM5_slave_num 5</pre>					
56						
57	`define PWMCluster_err_sl	ave_num 6				
58						
59	`define PWM0Base 'h00	03_0000				
60	`define PWM0End 'h00	03_00FF				
61	`define PWM1Base 'h00	03_0100				
62	`define PWM1End 'h00	03_01FF				

🗙 Welcom	ne ≣ Soc.bsv ≣ Soc.defines ● 🗟	2 Settings		□ …
≡ Soc.de				
31	`define Err_slave_num 6		> uart_cluster	Aa ab $*$ No results $\land \downarrow \equiv \times$
32				
33	`define BootBase	'h0000_1000		
34	`define BootEnd	'h0000_2FFF		
35	`define UARTClusterBase	'h0001_1300_		
36	`define.UARTClusterEnd	·'h0001_1640		
37	`define SPIClusterBase	'h0002_0000		
38	`define SPIClusterEnd	'h0002_01FF		
39	`define PWMClusterBase	'h0003_0000		
40	`define PWMClusterEnd	'h0003_05FF		
41	`define MixedClusterBase	'h0004_0000		
42	`define MixedClusterEnd	'h0004_15FF		
43	`define EthBase	'h0004_4000		
44	`define EthEnd	'h0004_7FFF		
45				
46	// - PWM cluster			
47	`define PWMCluster_Num_Sl	aves 7		
48	`define PWMCluster_Num_Ma	sters 1		
49				
50	`define PWM0_slave_num 0			
51	`define PWM1_slave_num 1			
52	`define PWM2_slave_num 2			
53	`define PWM3_slave_num 3			
54	`define PWM4 slave num 4			
55	`define PWM5 slave num 5			
56				
57	`define PWMCluster_err_sl	ave_num 6		
58				
59	`define PWM0Base 'h00	03 0000		
60	`define PWM0End 'h00	03_00FF		

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

Welcome	□ …
≣ Soc.defines	
80 \downarrow wart_cluster Aa \Rightarrow No results $\uparrow \downarrow =$	×
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	
88 // - UART cluster	
89 `define UARTCluster_Num_Slaves 4	
90 `define UARTCluster_Num_Masters 1	
92 `define UARTO_slave_num 0	
93 `define UART1_slave_num 1	
94 `define UART2_slave_num 2	
95 `define UARTCluster_err_slave_num 3	
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 T2C1 slave num 1	

From "4" to "5".

🗙 Welcom	ne E Soc.bsv E Soc.defines ● Z Settings	□ …
≡ Soc.de		
/9	derine Spicluster_err_stave_num 2	> uart_cluster Aa ab * No results $\uparrow \downarrow = x$
80		
81	<pre>`define SPI0Base 'h0002_0000 `define SPI0Fad</pre>	
82	<pre>`define SPI0End 'h0002_00FF `define SPI0End 'h0002_0100</pre>	
83	<pre>`define SPI1Base 'h0002_0100 `define SPI1Ead</pre>	
84	<pre>`define SPI1End 'h0002_01FF (/`define SPI2Base h0002_0200</pre>	
85	<pre>//`define SPI2Base 'h0002_0200 //`define SPI2Ead</pre>	
86	//`define SPI2End 'h0002_02FF	
87		
88	// - UART cluster	
89	<pre>`define UARTCluster_Num_Slaves 5</pre>	
90	`define UARTCluster_Num_Masters 1	
91 92	define HADTO slave num O	
92	`define UART0_slave_num 0 `define UART1 slave num 1	
93 94	define UART2 slave num 2	
94 95		
95 96	`define UARTCluster_err_slave_num 3	
90 97	`define UART0Base 'h0001_1300	
98	define UARTOEnd 'h0001 1340	
99	define UARTIBase 'h0001 1400	
100	define UARTIEnd 'h0001 1440	
101	define UART2Base 'h0001 1500	
101	define UART2End 'h0001 1540	
102		
104	// - Mixed Cluster	
105	define MixedCluster Num Slaves 7	
106	`define MixedCluster_Num_Masters 1	
100		
108	`define I2C0 slave num 0	
100	<pre>`define I2C1 slave num 1</pre>	

Step 3 (Add slave number for the new UART IP):

Add a slave number for the new slave UART. i.e. Change the

X Welcome	□ …
E Soc.defines	
/9 define Spiciuster_err_stave_num 2 > uart_duster Aa ab .* No results ↑ ↓	$\equiv \times$
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.UARTO_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 T2C1 slave num 1	

🗙 Welcon	ne E Soc.bsv E uart_cluster.bsv ● E Soc.defines X Z Settings		□ …
≣ Soc.de	fines		
81	`define SPI0Base	> uart0 Aa ab * 1 of 3	$\uparrow \downarrow = \times$
82	`define SPI0End 'h0002_00FF		
83	`define SPI1Base 'h0002_0100		
84	`define SPI1End		
85	<pre>//`define SPI2Base 'h0002_0200</pre>		
86	//`define SPI2End 'h0002_02FF		
87			
88	// - UART cluster		
89	<pre>`define UARTCluster_Num_Slaves 5</pre>		
90	<pre>`define UARTCluster_Num_Masters 1</pre>		
91			
92	`define UART0_slave_num 0		
93	<pre>`define UART1_slave_num 1</pre>		
94	`define UART2_slave_num 2		
95	`define UART3_slave_num 3		
96	<pre>`define UARTCluster_err_slave_num 4</pre>		
97			
98	`define UART0Base		
99	`define UART0End		
100	`define UART1Base 'h0001_1400		
101	`define UART1End		
102	`define UART2Base		
103	`define UART2End		
104	`define UART3Base 'h0001_1600		
105	`define UART3End		
106			
107	// - Mixed Cluster		
108	<pre>`define MixedCluster_Num_Slaves 7</pre>		
109	`define MixedCluster_Num_Masters 1		
110			

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

🗙 Welcome	≣ Soc.bsv	≡ Soc.defines ●						□ …
≡ Soc.defin	es							
79 80	derine SPic	luster_err_	stave_num	2	> uart_cluster	Aa _ab* No results	↑ ↓ ≡	×
	define SPI0	Bace 'b	0002 0000					
	define SPI0		0002_0000 0002_00FF					
	define SPI1		0002_0001					
	define SPI1		0002_0100 0002 01FF					
	//`define SP:		'h0002 020	Θ				
-	//`define SP:		'h0002_02F					
87	y actine bi.		110002_021	•				
	// - UART clu	uster						
-	define UART		Slaves 5					
90	define UART							
91								
	define UART	0 slave num	Θ					
	define UART							
	define UART							
	define UART							
	define UART			4				
97								
98	define UART	9Base∙∙∙∙'	h0001_1300					
99	define UART	9End · · · · · · '	h0001_1340					
100	define UART	1Base·····	h0001_1400					-
101	define UART	1End · · · · · · '	h0001_1440					
102	define UART2	2Base·····	h0001_1500					
103	define UART2	2End · · · · · · '	h0001_1540					
104								
105 ,	// - Mixed C	luster						
106	define Mixed	dCluster_Nu	m_Slaves 7					
	define Mixed	dCluster_Nu	m_Masters	1				
108								
109	define T2C0	slave num	0					

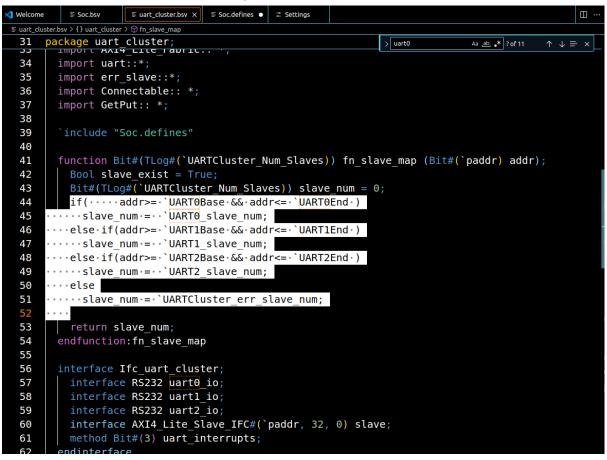
i.e . Add memory map

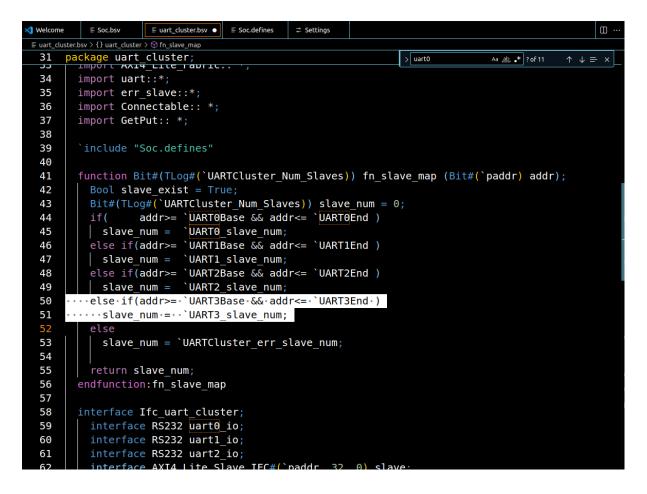
🗙 Welcom	e ≣ Soc.bsv	E Soc.defines ●					Π
≡ Soc.de		_ bockdennes •	a seconds				
/9	derine SPic	tuster_err_	scave_num	2	> uart_cluster	Aa _ab_ * No results	$\wedge \downarrow = x$
80					, <u></u>	Horesaus	- + + - ×
81	`define SPI0		10002_0000				
82	`define SPI0		10002_00FF				
83	`define SPI1		10002_0100				
84	`define SPI1		10002_01FF				
85	<pre>//`define SP</pre>		'h0002_020				
86	<pre>//`define SP</pre>	I2End	'h0002_02F	F			
87	-						
88	// - UART cl		-				
89	`define UART						
90	`define UART	Cluster_Num	n_Masters 1				
91			-				
92	`define UART						
93	`define UART						
94	`define UART	_					
95	`define UART						
96	`define UART	Cluster_err	_slave_num	4			
97							
98	\define UART		h0001_1300				
99	`define UART		h0001_1340				
100	`define UART		h0001_1400				
101	`define UART		h0001_1440				
102	`define UART		h0001_1500				
103	`define UART `define∙UART		h0001_1540				
104			_				
105	`define UART	SEIIU · · · · ·	10001_1040				
106 107	// Mixed C	luctor					
107	<pre>// - Mixed C `define Mixe</pre>						
	`define Mixe						

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





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

🗙 Welcome	E Soc.bsv E uart_cluster.bsv ● E Soc.defines Z Settings
≡ uart_clu	ster.bsv > () uart_cluster > +0 lfc_uart_cluster
31	package uart_cluster; \Rightarrow varto $A_{a} \stackrel{\text{de}}{\longrightarrow} *$ of 11 $\uparrow \downarrow \equiv \times$
41	<pre>function Bit#(TLog#(`UARTCluster_Num_Slaves)) fn_slave_map (Bit#(`paddr) addr);</pre>
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	<pre>interface AXI4_Lite_Slave_IFC#(`paddr, 32, 0) slave;</pre>
63	<pre>method Bit#(3) uart_interrupts;</pre>
64	endinterface
65	
66	(*synthesize*)
67	<pre>module mkuart(Ifc_uart_axi4lite#(32, 32, 0, 16));</pre>
68	<pre>let core_clock<-exposeCurrentClock;</pre>
69	<pre>let core_reset<-exposeCurrentReset;</pre>
70	let ifc();
71	<pre>mkuart_axi4lite#(core_clock, core_reset, 163, 0, 0) _temp(ifc);</pre>
72	return ifc;
73	endmodule
74	
75	(*synthesize*)
76	<pre>module mkuart_cluster(Ifc_uart_cluster);</pre>
77	<pre>let curr_clk<- exposeCurrentClock;</pre>
78	<pre>let curr_reset <- exposeCurrentReset;</pre>
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)

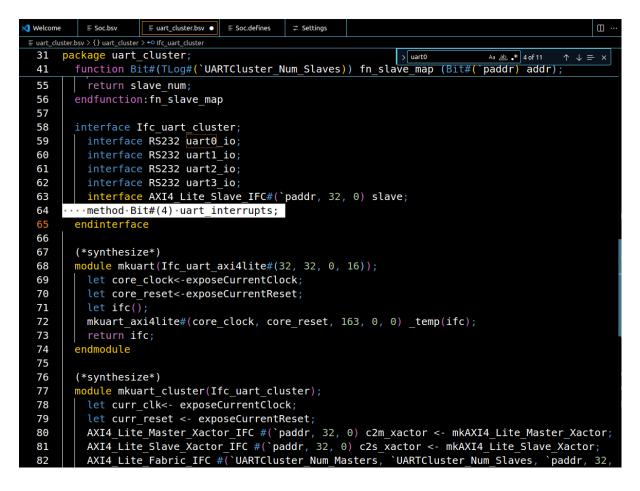
🗙 Welcome	E Soc.bsv E uart_cluster.bsv ● E Soc.defines Z Settings	
	ster.bsv > {} uart_cluster > +0 Ifc_uart_cluster	
31	package uart_cluster; y arto An $(A_{A}, (A_{A}, A_{A}))$ $(A_{A}, (A_{A}, A_{A}))$ $(A_{A}, (A_{A}, A_{A}))$	
41	<pre>function Bit#(TLog#(`UARTCluster_Num_Slaves)) fn_slave_map (Bit#(`paddr) addr);</pre>	
55	return slave_num;	
56	endfunction:fn_slave_map	
57		
58	<pre>interface Ifc_uart_cluster;</pre>	
59	interface RS232 uart0_io;	
60	interface RS232 uart1_io;	
61	<pre>interface RS232 uart2_io;</pre>	
62	<pre>interface.RS232.uart3_io;</pre>	
63	<pre>interface AXI4_Lite_Slave_IFC#(`paddr, 32, 0) slave;</pre>	
64	<pre>method Bit#(3) uart_interrupts;</pre>	
65	endinterface	
66		
67	(*synthesize*)	
68	<pre>module mkuart(Ifc_uart_axi4lite#(32, 32, 0, 16));</pre>	
69	<pre>let core_clock<-exposeCurrentClock;</pre>	
70	<pre>let core_reset<-exposeCurrentReset;</pre>	
71	<pre>let ifc();</pre>	
72	<pre>mkuart_axi4lite#(core_clock, core_reset, 163, 0, 0) _temp(ifc);</pre>	
73	return ifc;	
74	endmodule	
75		
76		
77	<pre>module mkuart_cluster(Ifc_uart_cluster);</pre>	
78	<pre>let curr_clk<- exposeCurrentClock;</pre>	
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):

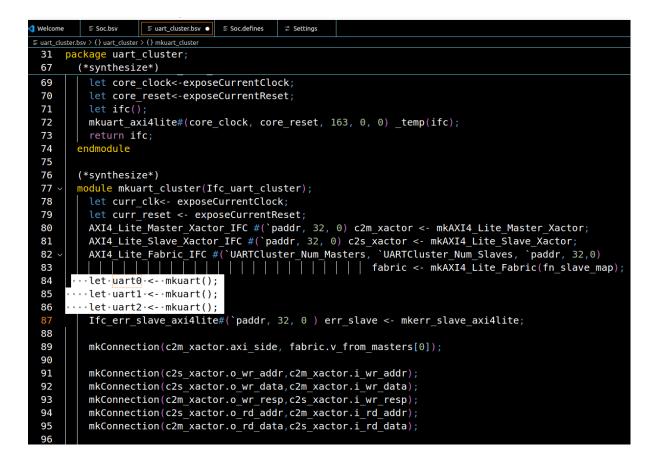
Add one more bit for the new UART Interrupt i.e. change the code from **Bit#(3)**

🗙 Welcome	
	ister.bsv > {} uart_cluster > +0 Ifc_uart_cluster
	package uart_cluster; \downarrow uart0 Aa $\frac{ab}{4}$ * 4of 11 $\uparrow \downarrow \equiv x$
41	<pre>function Bit#(TLog#(`UARTCluster_Num_Slaves)) fn_slave_map (Bit#(`paddr) addr);</pre>
55	return slave_num;
56	endfunction:fn_slave_map
57	
58	interface Ifc_uart_cluster;
59	<pre>interface RS232 uart0_io;</pre>
60	<pre>interface RS232 uart1_io;</pre>
61	<pre>interface RS232 uart2_io;</pre>
62	<pre>interface RS232 uart3_io;</pre>
63	<pre>interface AXI4_Lite_Slave_IFC#(`paddr, 32, 0) slave;</pre>
64	<pre>method.Bit#(3).uart_interrupts;</pre>
65	endinterface
66	
67	(*synthesize*)
68	<pre>module mkuart(Ifc_uart_axi4lite#(32, 32, 0, 16));</pre>
69	<pre>let core_clock<-exposeCurrentClock;</pre>
70	<pre>let core_reset<-exposeCurrentReset;</pre>
71	<pre>let ifc();</pre>
72	<pre>mkuart_axi4lite#(core_clock, core_reset, 163, 0, 0) _temp(ifc);</pre>
73	return ifc;
74 75	endmodule
75	(*synthesize*)
78	<pre>module mkuart cluster(Ifc uart cluster);</pre>
78	let curr clk<- exposeCurrentClock;
78	let curr reset <- exposeCurrentReset;
80	AXI4_Lite_Master_Xactor_IFC #(`paddr, 32, 0) c2m_xactor <- mkAXI4_Lite_Master_Xactor;
81	AXI4_Lite_Master_Aactor_IfC #(`paddr, 32, 0) c2m_Aactor <- mkAXI4_Lite_Master_Aactor;
82	AXI4_Lite Fabric IFC #(`UARTCluster Num Masters, `UARTCluster Num Slaves, `paddr, 32,
02	

to Bit#(4)



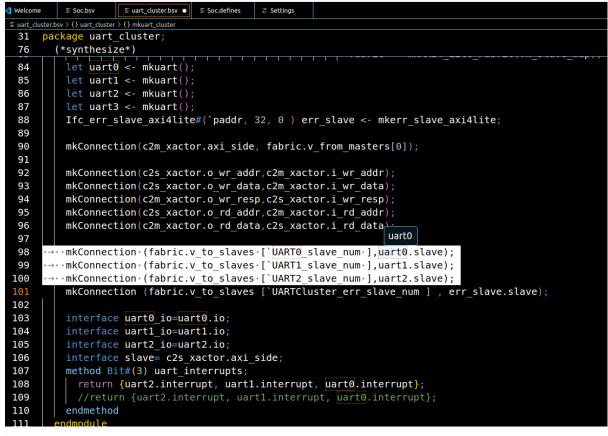
Setp 4 (Add a UART module instance by assigning it to a new interface instance): Add one more instance for the UART interface i.e. change the code from



Welcome				
≣ uart_clu	ster.bsv > () uart_cluster > () mkuart_cluster			
31	31 package uart_cluster;			
67	(*synthesize*)			
69	<pre>let core clock<-exposeCurrentClock;</pre>			
70	<pre>let core_reset<-exposeCurrentReset;</pre>			
71	<pre>let ifc();</pre>			
72	<pre>mkuart_axi4lite#(core_clock, core_reset, 163, 0, 0) _temp(ifc);</pre>			
73	return ifc;			
74	endmodule			
75				
76	(*synthesize*)			
77	<pre>module mkuart_cluster(Ifc_uart_cluster);</pre>			
78	<pre>let curr_clk<- exposeCurrentClock;</pre>			
79	<pre>let curr_reset <- exposeCurrentReset;</pre>			
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				
83	3			
84	<pre>let uart0 <- mkuart();</pre>			
85	<pre>let uart1 <- mkuart();</pre>			
86	<pre>let uart2 <- mkuart();</pre>			
87	<pre>let.uart3.<mkuart();< pre=""></mkuart();<></pre>			
88	<pre>Ifc_err_slave_axi4lite#(`paddr, 32, 0) err_slave <- mkerr_slave_axi4lite;</pre>			
89				
90	<pre>mkConnection(c2m_xactor.axi_side, fabric.v_from_masters[0]);</pre>			
91				
92	<pre>mkConnection(c2s_xactor.o_wr_addr,c2m_xactor.i_wr_addr);</pre>			
93	<pre>mkConnection(c2s_xactor.o_wr_data,c2m_xactor.i_wr_data);</pre>			
94	<pre>mkConnection(c2m_xactor.o_wr_resp,c2s_xactor.i_wr_resp);</pre>			
95	<pre>mkConnection(c2s_xactor.o_rd_addr,c2m_xactor.i_rd_addr);</pre>			
96	<pre>mkConnection(c2m xactor.o rd data,c2s xactor.i rd data);</pre>			

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

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



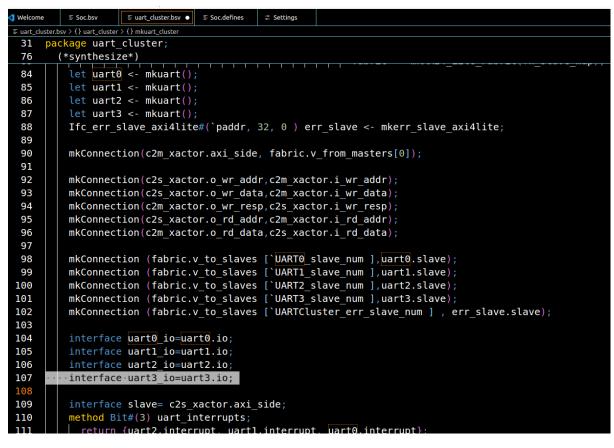
```
≣ uart_cluster.bsv ● ≡ Soc.defines ≠ Settings
🜖 Welcome
         ≣ Soc.bsv
≡ uart_cluster.bsv > { } uart_cluster > { } mkuart_clust
 31 package uart_cluster;
 76
         (*synthesize*)
           let uart0 <- mkuart();</pre>
 84
 85
           let uart1 <- mkuart();</pre>
           let uart2 <- mkuart();</pre>
 86
 87
           let uart3 <- mkuart();</pre>
          Ifc_err_slave_axi4lite#(`paddr, 32, 0 ) err_slave <- mkerr_slave_axi4lite;</pre>
 88
 89
          mkConnection(c2m_xactor.axi_side, fabric.v_from_masters[0]);
 90
 91
 92
           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);
 93
 94
           mkConnection(c2s_xactor.o_rd_addr,c2m_xactor.i_rd_addr);
 95
 96
           mkConnection(c2m_xactor.o_rd_data,c2s_xactor.i_rd_data);
 97
 98
           mkConnection (fabric.v_to_slaves [`UART0_slave_num ],uart0.slave);
          mkConnection (fabric.v to slaves [`UART1_slave_num ],uart1.slave);
 99
          mkConnection (fabric.v_to_slaves [`UART2_slave_num ],uart2.slave);
100
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;
           interface slave= c2s_xactor.axi_side;
107
108
          method Bit#(3) uart interrupts;
109
             return {uart2.interrupt, uart1.interrupt, uart0.interrupt};
110
             //return {uart2.interrupt, uart1.interrupt, uart0.interrupt};
               ethod
```

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	E Soc.bsv E uart_cluster.bsv ● E Soc.defines			
≡ uart_cluster	.bsv > () uart_cluster > () mkuart_cluster			
31 pa	ackage uart_cluster;			
76	(*synthesize*)			
84	let uart0 <- mkuart();			
85	<pre>let uart1 <- mkuart();</pre>			
86				
87	let uart3 <- mkuart();			
88	<pre>Ifc_err_slave_axi4lite#(`paddr, 32, 0) err_slave <- mkerr_slave_axi4lite;</pre>			
89				
90	<pre>mkConnection(c2m_xactor.axi_side, fabric.v_from_masters[0]);</pre>			
91				
92	<pre>mkConnection(c2s_xactor.o_wr_addr,c2m_xactor.i_wr_addr);</pre>			
93				
94				
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	97			
98	<pre>mkConnection (fabric.v_to_slaves [`UART0_slave_num],uart0.slave);</pre>			
99	<pre>mkConnection (fabric.v_to_slaves [`UART1_slave_num],uart1.slave);</pre>			
100	<pre>mkConnection (fabric.v_to_slaves [`UART2_slave_num],uart2.slave);</pre>			
101	<pre>mkConnection (fabric.v_to_slaves [`UART3_slave_num],uart3.slave);</pre>			
102	<pre>mkConnection (fabric.v_to_slaves [`UARTCluster_err_slave_num] , err_slave.slave);</pre>			
103				
104 · ·	104 ····interface·uart0_io=uart0.io;			
105 · ·	<pre>interface.uart1_io=uart1.io;</pre>			
106 · ·	<pre>interface.uart2_io=uart2.io;</pre>			
107	<pre>interface slave= c2s_xactor.axi_side;</pre>			
108	<pre>method Bit#(3) uart_interrupts;</pre>			
109	<pre>return {uart2.interrupt, uart1.interrupt, uart0.interrupt};</pre>			
110	<pre>//return {uart2.interrupt, uart1.interrupt, uart9.interrupt};</pre>			
111	ul endmethod			

to



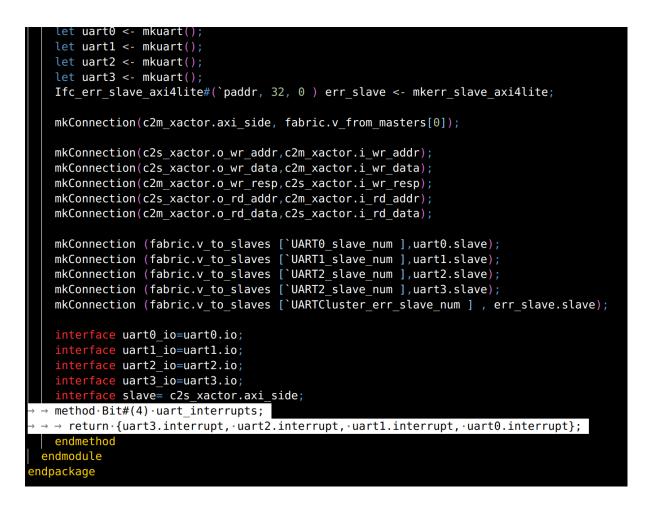
Step 7 (Add one more UART Interrupt bit to take the same to PLIC):

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

 Welcome
 E Soc.bsv
 E uart_cluster.bsv ●
 E Soc.defines

 E uart_cluster.bsv > {} uart_cluster > {} mkuart_cluster > {} uart_interrupts
 🛱 Settings d Welcome 31 package uart_cluster; (*synthesize*) Let uart2 <- mкuart(); 76 86 87 let uart3 <- mkuart();</pre> Ifc_err_slave_axi4lite#(`paddr, 32, 0) err_slave <- mkerr_slave_axi4lite;</pre> 88 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); 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 96 mkConnection(c2m_xactor.o_rd_data,c2s_xactor.i_rd_data); 97 98 mkConnection (fabric.v_to_slaves [`UART0_slave_num],uart0.slave); mkConnection (fabric.v to slaves [`UART1 slave num],uart1.slave); 99 mkConnection (fabric.v_to_slaves [`UART2_slave_num],uart2.slave); 100 mkConnection (fabric.v_to_slaves [`UART3_slave_num],uart3.slave); mkConnection (fabric.v_to_slaves [`UARTCluster_err_slave_num] , err_slave.slave); 101 102 103 104 interface uart0_io=uart0.io; interface uart1_io=uart1.io; interface uart2_io=uart2.io; 105 106 interface uart3_io=uart3.io; 107 108 109 interface slave= c2s_xactor.axi_side method Bit#(3) uart_interrupts; 110 111→ return {uart2.interrupt, uart1.interrupt, uart0.interrupt}; //return {uart2.interrupt, uart1.interrupt, uart0.interrupt}; 112 113 endmethod

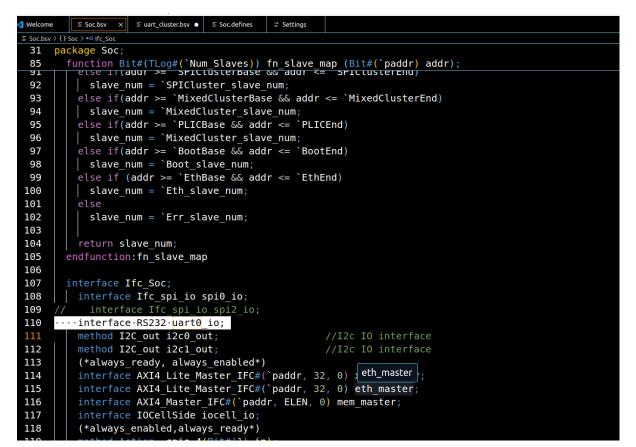
То

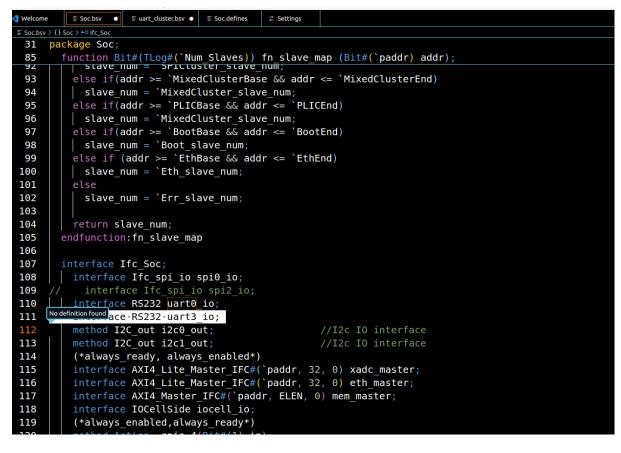


3. Make the following changes in the "soc.bsv".

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





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

Assign the interface with UART cluster by changing the code from

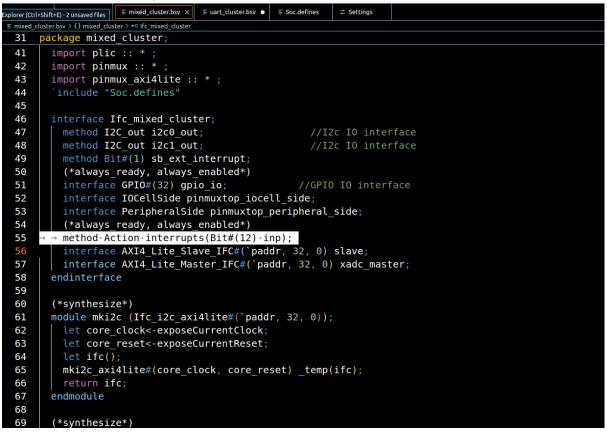
Welcome	E Soc.bsv ● E uart_cluster.bsv ● E Soc.defines					
	() Soc> () mkSoc					
•	wackage Soc;					
275	(*synthesize*)					
631						
632	<pre>method gpio_20_outen = mixed_cluster.gpio_io.gpio_out_en[20];</pre>					
633	<pre>method gpio_21_outen = mixed_cluster.gpio_io.gpio_out_en[21];</pre>					
634	<pre>method gpio_22_outen = mixed_cluster.gpio_io.gpio_out_en[22];</pre>					
635	<pre>method gpio_23_outen = mixed_cluster.gpio_io.gpio_out_en[23];</pre>					
636	<pre>method gpio_24_outen = mixed_cluster.gpio_io.gpio_out_en[24];</pre>					
637	<pre>method gpio_25_outen = mixed_cluster.gpio_io.gpio_out_en[25];</pre>					
638	<pre>method gpio_26_outen = mixed_cluster.gpio_io.gpio_out_en[26];</pre>					
639	<pre>method gpio_27_outen = mixed_cluster.gpio_io.gpio_out_en[27];</pre>					
640	<pre>method gpio_28_outen = mixed_cluster.gpio_io.gpio_out_en[28];</pre>					
641	<pre>method gpio_29_outen = mixed_cluster.gpio_io.gpio_out_en[29];</pre>					
642	<pre>method gpio_30_outen = mixed_cluster.gpio_io.gpio_out_en[30];</pre>					
643	<pre>method gpio_31_outen = mixed_cluster.gpio_io.gpio_out_en[31];</pre>					
644	interface spi0_io = spi_cluster.sp					
645 /	/ interface spi1_io = spi_cluster. uart0_io					
646 •	<pre>interface.uart0_io = uart_cluster.uart0_io;</pre>					
647	<pre>method i2c0_out = mixed_cluster.i2c0_out; //I2c I0 interface</pre>					
648	<pre>method i2c1_out = mixed_cluster.i2c1_out; //I2c I0 interface</pre>					
649	<pre>interface iocell_io = mixed_cluster.pinmuxtop_iocell_side; //GPI0 I0 interface</pre>					
650	<pre>interface xadc_master = mixed_cluster.xadc_master;</pre>					
651	<pre>interface eth_master = slow_fabric.v_to_slaves[`Eth_slave_num];</pre>					
652	<pre>interface mem_master = fabric.v_to_slaves [`Memory_slave_num];</pre>					
653	<pre>method Action ext_interrupts(Bit#(3) i);</pre>					
654	wr_ext_interrupts <= i;					
655	endmethod					
656	5 `ifdef rtldump					
657	<pre>interface io_dump= eclass.io_dump;</pre>					
658	`endif					

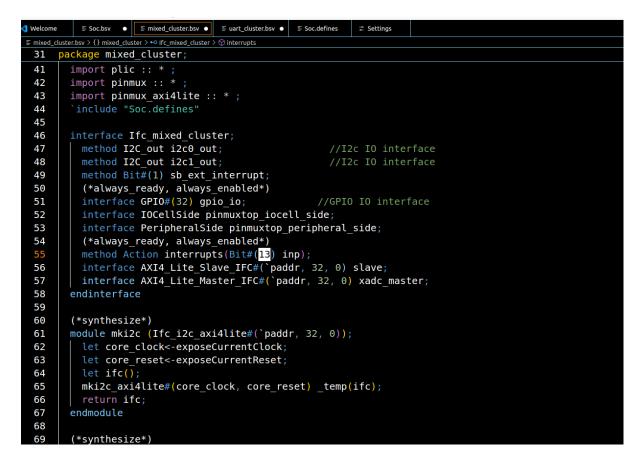
Welcome	E Soc.bsv ● E uart_cluster.bsv ● E Soc.defines [*] Settings		
≣ Soc.bsv > {	⊊ Soc.bsv > () Soc > () mkSoc		
31 pa	31 package Soc;		
275	(*synthesize*)		
631	<pre>method gpio_19_outen = mixed_cluster.gpio_io.gpio_out_en[19];</pre>		
632	<pre>method gpio_20_outen = mixed_cluster.gpio_io.gpio_out_en[20];</pre>		
633	<pre>method gpio_21_outen = mixed_cluster.gpio_io.gpio_out_en[21];</pre>		
634	<pre>method gpio_22_outen = mixed_cluster.gpio_io.gpio_out_en[22];</pre>		
635	<pre>method gpio_23_outen = mixed_cluster.gpio_io.gpio_out_en[23];</pre>		
636	<pre>method gpio_24_outen = mixed_cluster.gpio_io.gpio_out_en[24];</pre>		
637	<pre>method gpio_25_outen = mixed_cluster.gpio_io.gpio_out_en[25];</pre>		
638	<pre>method gpio_26_outen = mixed_cluster.gpio_io.gpio_out_en[26];</pre>		
639	<pre>method gpio_27_outen = mixed_cluster.gpio_io.gpio_out_en[27];</pre>		
640	<pre>method gpio_28_outen = mixed_cluster.gpio_io.gpio_out_en[28];</pre>		
641	<pre>method gpio_29_outen = mixed_cluster.gpio_io.gpio_out_en[29];</pre>		
642	<pre>method gpio_30_outen = mixed_cluster.gpio_io.gpio_out_en[30];</pre>		
643	<pre>method gpio_31_outen = mixed_cluster.gpio_io.gpio_out_en[31];</pre>		
644	<pre>interface spi0_io = spi_cluster.spi0_io;</pre>		
645 //	interface spil_io = spi_cluster.spil_io;		
646 • •	<pre>interface.uart0_io.=.uart_cluster.uart0_io;</pre>		
647 • •	<pre>interface.uart3_io.=.uart_cluster.uart3_io;</pre>		
648	<pre>method i2c0 out = mixed cluster.i2c0 out; //I2c I0 interface</pre>		
649	<pre>method i2c1_out = mixed_cluster.i2c1_out; //I2c I0 interface</pre>		
650	<pre>interface iocell_io = mixed_cluster.pinmuxtop_iocell_side; //GPI0 I0 interface</pre>		
651	<pre>interface xadc_master = mixed_cluster.xadc_master;</pre>		
652	<pre>interface eth_master = slow_fabric.v_to_slaves[`Eth_slave_num];</pre>		
653	interface mem master = fabric.v to slaves [`Memory slave num];		
654	<pre>method Action ext_interrupts(Bit#(3) i);</pre>		
655	wr_ext_interrupts <= i;		
656	656 endmethod		
657	7 ['] ifdef rtldump		
658	<pre>interface io_dump= eclass.io_dump;</pre>		

4. Make the following changes in the "mixed_cluster.bsv".

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

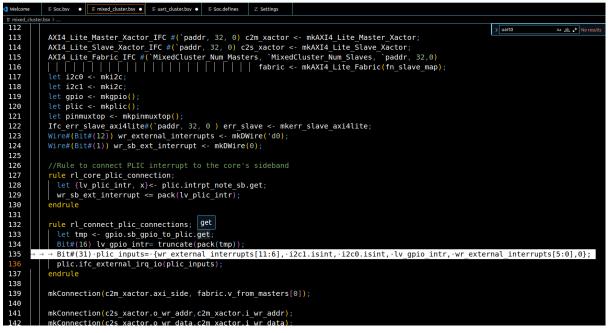
Do the following changes in mixed_cluster.bsv i.e. increase the interrupt bits from 12 to 13. Change the code from

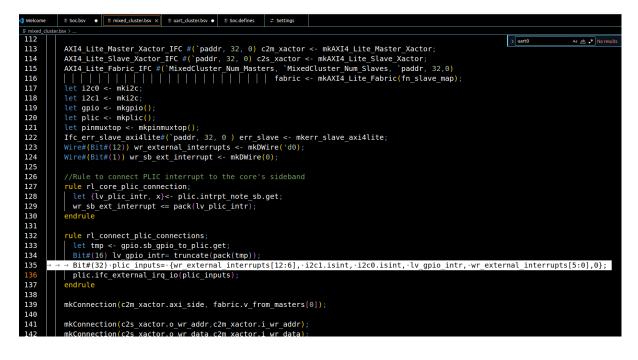




Step 2 (interrupts passed as input to PLIC):

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

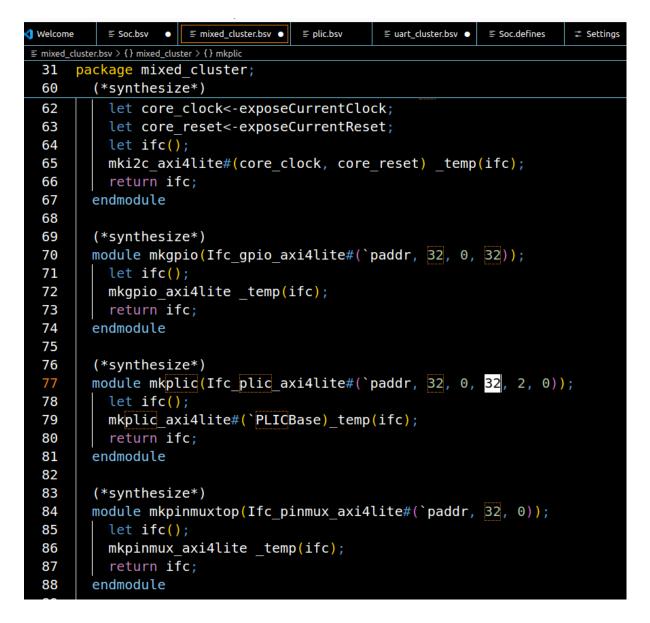




Step 3: Add one more interrupt to PLIC module

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

d Welcome	E Soc.bsv E mixed_cluster.bsv × E plic.bsv E uart_cluster.bsv ● = soc.uennes ~ settings			
	<pre>mixed_cluster.bsv > {} mixed_cluster > {} mkplic</pre>			
31 60	31 <pre>package mixed_cluster;</pre>			
	(*synthesize*)			
62	<pre>let core_clock<-exposeCurrentClock;</pre>			
	63 let core_reset<-exposeCurrentReset;			
64	<pre>let ifc();</pre>			
65	<pre>mki2c_axi4lite#(core_clock, core_reset) _temp(ifc);</pre>			
66	return ifc;			
67				
68	(touthorizot)			
69 70	(*synthesize*)			
70 71	<pre>module mkgpio(Ifc_gpio_axi4lite#(`paddr, 32, 0, 32)); let ifc();</pre>			
71	<pre>mkgpio axi4lite temp(ifc);</pre>			
72	return ifc;			
73	endmodule			
74				
75	(*synthesize*)			
	<pre>* module mkplic(Ifc plic axi4lite#(`paddr, 32, 0, 31, 2, 0));</pre>			
78	let ifc();			
79	<pre>mkplic axi4lite#(`PLICBase) temp(ifc);</pre>			
80	return ifc;			
81	endmodule			
82				
83	(*synthesize*)			
84	<pre>module mkpinmuxtop(Ifc pinmux axi4lite#(`paddr, 32, 0));</pre>			
85	<pre>let ifc();</pre>			
86	<pre>mkpinmux axi4lite temp(ifc);</pre>			
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

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;</pre> endmethod interface pinmuxtop_iocell_side = pinmuxtop.pinmuxaxi4lite_iocell_side;

```
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 [`GPI0_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);
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, i2c1 sda, inout inout i2c1 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





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
<pre>#set_property -dict { PACKAGE_PIN U12 IOSTANDARD LVCMOS33 } [get_ports { spi1_nss }]; #I0_L20P_T3_A08_D24_14 Sch=jc_p[1]</pre>
<pre>#set_property -dict { PACKAGE_PIN V12 IOSTANDARD LVCMOS33 } [get_ports { spi1_mosi }]; #I0_L20N_T3_A07_D23_14 Sch=jc_n[1]</pre>
<pre>#set_property -dict { PACKAGE_PIN V10 IOSTANDARD LVCMOS33 } [get_ports { spi1_miso }]; #I0_L21P_T3_DQS_14 Sch=jc_p[2]</pre>
<pre>#set_property -dict { PACKAGE_PIN U14 IOSTANDARD LVCMOS33 } [get_ports { spi1_sclk] }]; #I0_L22P_T3_A05_D21_14 Sch=jc_p[3]</pre>
USB-UART Interface
<pre>set_property - dict - { · PACKAGE_PIN D10 · · · IOSTANDARD LVCMOS33 · } · [get_ports · { · uart0_SOUT · }]; · #I0_L19N_T3_VREF_16 · Sch=uart_rxd_out</pre>
<pre>set_property -dict { PACKAGE_PIN A9 · · · IOSTANDARD LVCMOS33 } [get_ports { uart0_SIN }]; #I0_L14N_T2_SRCC_16 Sch=uart_txd_in</pre>
ChipKit Outer Digital Header
<pre>set_property -dict { PACKAGE_PIN V15 IOSTANDARD LVCMOS33 } [get_ports { io7_cell }]; #IO_L16P_T2_CSI_B_14 Sch=ck_io[0]</pre>
<pre>set_property PULLDOWN true [get_ports { io7_cell }];</pre>

to

Pmod Header JC
<pre>#set_property -dict { PACKAGE_PIN U12 IOSTANDARD LVCM0S33 } [get_ports { spi1_nss }]; #I0_L20P_T3_A08_D24_14 Sch=jc_p[1]</pre>
<pre>#set_property -dict { PACKAGE_PIN V12 IOSTANDARD LVCM0S33 } [get_ports { spi1_mosi }]; #I0_L20N_T3_A07_D23_14 Sch=jc_n[1]</pre>
<pre>#set_property -dict { PACKAGE_PIN V10 IOSTANDARD LVCMOS33 } [get_ports { spi1_miso }]; #IO_L21P_T3_DQS_14 Sch=jc_p[2]</pre>
<pre>#set_property -dict { PACKAGE_PIN U14 IOSTANDARD LVCM0S33 } [get_ports { spi1_sclk] }]; #I0_L22P_T3_A05_D21_14 Sch=jc_p[3]</pre>
USB-UART Interface
<pre>set_property - dict { PACKAGE_PIN D10 · · IOSTANDARD LVCMOS33 } [get_ports { uart0_SOUT }]; #IO_L19N_T3_VREF_16 · Sch=uart_rxd_out</pre>
<pre>set_property -dict { PACKAGE_PIN A9 · · · IOSTANDARD LVCMOS33 } [get_ports { uart0_SIN }]; #IO_L14N_T2_SRCC_16 Sch=uart_txd_in _</pre>
<pre>set_property - dict { PACKAGE_PIN xxx · IOSTANDARD LVCMOS33 } [get_ports { uart3_SOUT }]; #IO_L19N_T3_VREF_16 · Sch=uart_rxd_out_</pre>
<pre>set_property -dict { PACKAGE_PIN yyy ··· IOSTANDARD LVCMOS33 } [get_ports { uart3_SIN }]; #IO_L14N_T2_SRCC_16 Sch=uart_txd_in</pre>
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 <u>https://gitlab.com/shaktiproject/cores/crypto-box</u>

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

Add the following command in Makefile

<pre>@cp \${BS_VERILOG_LIB}/ResetEither.v \${VERILOGDIR}</pre>
<pre>@cp \${BS_VERILOG_LIB}/MakeReset0.v \${VERILOGDIR}</pre>
<pre>@cp \${BS_VERILOG_LIB}/SyncReset0.v \${VERILOGDIR}</pre>
<pre>@cp \${BS_VERILOG_LIB}/ClockInverter.v \${VERILOGDIR}</pre>
<pre>@cp \${BS_VERILOG_LIB}/SyncFIF01.v \${VERILOGDIR}</pre>
<pre>@cp \${BS_VERILOG_LIB}/RevertReg.v \${VERILOGDIR}</pre>
<pre>@cp \${BS_VERILOG_LIB}/BRAM1.v \${VERILOGDIR}</pre>
<pre>@cp \${BS_VERILOG_LIB}/BRAM2.v \${VERILOGDIR}</pre>
<pre>@cp ./common_verilog/bram_1rw.v \${VERILOGDIR}</pre>
<pre>@cp \${BS_VERILOG_LIB}/FIF020.v \${VERILOGDIR}</pre>
<pre>@cp ./common_verilog/bram_2rw.v \${VERILOGDIR}</pre>
<pre>@cp common_verilog/signedmul.v \${VERILOGDIR}</pre>
<pre>@cp \${BS_VERILOG_LIB}/SyncRegister.v \${VERILOGDIR}</pre>

Hierarchy

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

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



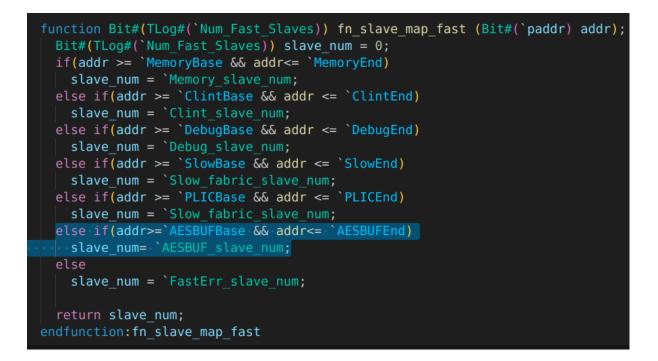
All the Memory mapping details will be in the **Soc.defines** file.

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	GPI0Base	'h0004_0100
`define	GPI0End	'h0004_01FF
`define	XADCBase	'h0004_1000
`define	XADCEnd	'h0004_13FF
`define	I2C1Base	'h0004_1400
`define	I2C1End	'h0004_14FF
`define	PinmuxBase	e 'h0004_1500
`define	PinmuxCon	figReg 'h0004_1510
`define	PinmuxEnd	'h0004_15FF
`define	PLICBase	'h0C00_0000
`define	PLICEnd	'h0C01_001F
`define	AESBUFBase	e '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),

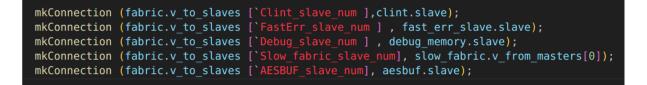


Declare the AES module and pass the required parameters like address width, data width, user width, clock and reset,

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

Ifc_pwm_cluster pwm_cluster <- mkpwm_cluster;</pre> Ifc uart cluster uart cluster <- mkuart cluster;</pre> Ifc_spi_cluster spi_cluster <- mkspi_cluster;</pre> Ifc_mixed_cluster mixed_cluster <- mkmixed_cluster;</pre> 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;</pre> Ifc_bram_axi4lite#(`paddr, 32, 0, 13) boot <- mkbram_axi4lite('h1000, "boot.mem", "Boot");</pre> Wire#(Bit#(3)) wr_ext_interrupts <- mkWire();</pre>

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,

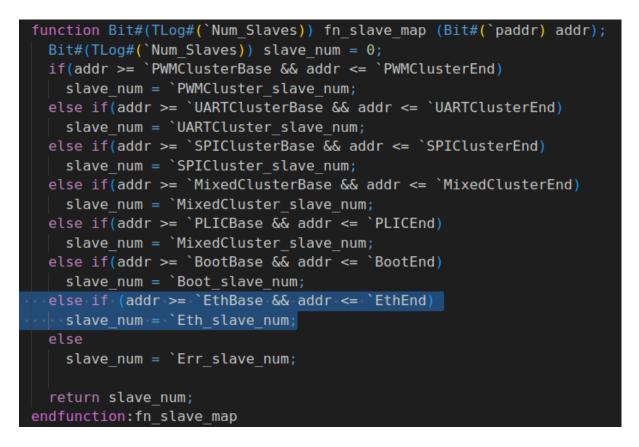


This completes the integrating fast IP in SP2020 to the core.

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.

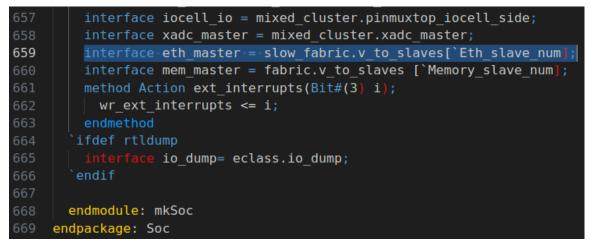
In Soc.bsv file, add slave_num for the new slave



Declare eth_master interface:

<pre>interface Ifc_Soc;</pre>
<pre>interface Ifc_spi_io spi0_io;</pre>
<pre>// interface Ifc_spi_io spi2_io;</pre>
<pre>interface RS232 uart0_io;</pre>
<pre>method I2C_out i2c0_out; //I2c I0 interface</pre>
<pre>method I2C_out i2c1_out; //I2c I0 interface</pre>
(*always_ready, always_enabled*)
<pre>interface AXI4_Lite_Master_IFC#(`paddr, 32, 0) xadc_master;</pre>
<pre>interface AXI4_Lite_Master_IFC#(`paddr, 32, 0) eth_master;</pre>
<pre>interface AXI4_Master_IFC#(`paddr, ELEN, 0) mem_master;</pre>
<pre>interface IOCellSide iocell_io;</pre>

Connect eth_master interface to slow fabric.



In Soc.defines file,

26	`define	Num_Slaves 7
27	`define	PWMCluster_slave_num 0
28	`define	UARTCluster_slave_num 1
29	`define	SPICluster_slave_num 2
30	`define	MixedCluster_slave_num 3
31	`define	Boot_slave_num 4
32	`define	Eth_slave_num 5
33	`define	Err_slave_num 6

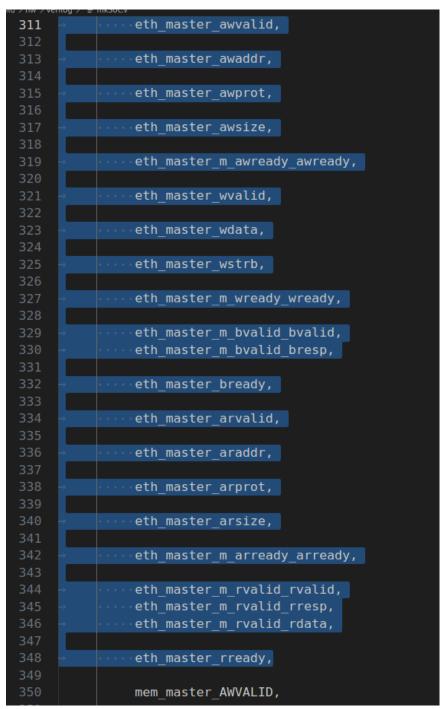
- 1. Increment value of Num_Slaves by 1.
- 2. Add `define Eth_slave_num *n* (replace *n* by the last slave number -1)
- 3. Change `define Err_slave_num *x* to `define Err_slave_num *x*+1 (*x* being the original value)

`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.

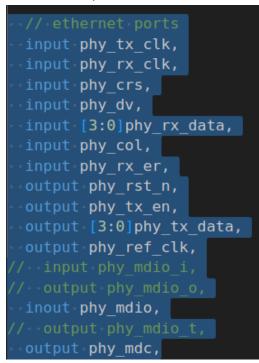
Generated mkSoc.v with newly added ports for ethernet slave:



- 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:



Initialise all wires:

```
wire [1 : 0] xadc master m rvalid rresp;
 wire [32-1 : 0] xadc master m rvalid rdata;
 wire xadc_master_rready;
 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	<pre>// Instantiating the C-class SoC//</pre>
546	mkSoc-core(
547	· · · // Main Clock and Reset to the SoC

Add the ports for new slave.

.locell_lo_lo20_cell_outen(lo20_cell_en),
// ETH connection
<pre>.eth_master_awvalid(eth_master_awvalid),</pre>
<pre>.eth_master_awaddr(eth_master_awaddr),</pre>
<pre>.eth_master_m_awready_awready(eth_master_m_awready_awready),</pre>
<pre>.eth_master_wvalid(eth_master_wvalid),</pre>
<pre>.eth_master_wdata(eth_master_wdata),</pre>
<pre>.eth_master_wstrb(eth_master_wstrb),</pre>
<pre>.eth_master_m_wready_wready(eth_master_m_wready_wready),</pre>
<pre>.eth_master_m_bvalid_bvalid(eth_master_m_bvalid_bvalid),</pre>
<pre>.eth_master_m_bvalid_bresp(eth_master_m_bvalid_bresp),</pre>
<pre>.eth_master_bready(eth_master_bready),</pre>
<pre>.eth_master_arvalid(eth_master_arvalid),</pre>
<pre>.eth_master_araddr(eth_master_araddr),</pre>
<pre>.eth_master_m_arready_arready(eth_master_m_arready_arready),</pre>
<pre>.eth_master_m_rvalid_rvalid(eth_master_m_rvalid_rvalid),</pre>
<pre>.eth_master_m_rvalid_rresp(eth_master_m_rvalid_rresp),</pre>
<pre>.eth_master_m_rvalid_rdata(eth_master_m_rvalid_rdata),</pre>
<pre>.eth_master_rready(eth_master_rready),</pre>
// 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	<pre></pre>
770	.s axi aresetn(~soc reset), // input wire s axi aresetn
771	ip2intc irpt(eth10 ip2intc irpt), // output wire ip2intc irpt
772	
773	.s axi awvalid (eth master awvalid),
774	.s axi awready (eth master m awready awready),
775	
776	
777	
778	.s axi wready (eth master m wready wready),
779	.s_axi_bresp (eth_master_m_bvalid_bresp),
780	
781	<pre>.s_axi_bready (eth_master_bready),</pre>
782	<pre></pre>
783	<pre></pre>
784	<pre>.s_axi_arready (eth_master_m_arready_arready),</pre>
785	<pre></pre>
786	<pre>.s_axi_rresp (eth_master_m_rvalid_rresp),</pre>
787	<pre>.s_axi_rvalid (eth_master_m_rvalid_rvalid),</pre>
788	.s_axi_rready (eth_master_rready),
789	<pre>phy_tx_clk(phy_tx_clk),//input.wire.phy_tx_clk</pre>
790	<pre></pre>
791	<pre>phy_crs(phy_crs),</pre>
792	phy_dv(phy_dv),
793	phy_rx_data(phy_rx_data), // input wire [3 : 0] phy_rx_data
794	<pre>phy_col(phy_col),//input.wire.phy_col</pre>
795	<pre>phy_rx_er(phy_rx_er),</pre>
796	<pre>phy_rst_n(phy_rst_n),//output.wire.phy_rst_nphy_tw_op(phy_tw_op)</pre>
797	phy_tx_en(phy_tx_en),//output.wire.phy_tx_en
798 799	<pre>phy_tx_data(phy_tx_data),//output.wire.[3.:.0].phy_tx_data</pre>
800	<pre>phy_mdio_i(phy_mdio_i),//.input.wire.phy_mdio_iphy_mdio_o(phy_mdio_o),//.output.wire.phy_mdio_o</pre>
801	
802	
803);
005	

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

123	#-SMSC·Ethernet-PHY
124	<pre>set_property -dict { PACKAGE_PIN D17 IOSTANDARD LVCMOS33 } [get_ports { phy_col }]; #IO_L16N_T2_A27_15 Sch=eth_col</pre>
125	<pre>set_property -dict { PACKAGE_PIN G14 ·· IOSTANDARD LVCMOS33 } [get_ports { phy_crs }]; #I0_L15N_T2_DQS_ADV_B_15 ·Sch=eth_crs</pre>
126	<pre>set_property -dict { PACKAGE_PIN F16 · IOSTANDARD LVCMOS33 } [get_ports { phy_mdc }]; #I0_L14N_T2_SRCC_15 Sch=eth_mdc</pre>
127	<pre>set_property -dict { PACKAGE_PIN K13 · IOSTANDARD LVCMOS33 } [get_ports { phy_mdio }]; #IO_L17P_T2_A26_15 Sch=eth_mdio</pre>
128	<pre>set_property - dict { PACKAGE_PIN G18 - · IOSTANDARD - LVCMOS33 - } [get_ports - { phy_ref_clk - }]; #I0_L22P_T3_A17_15</pre>
	Sch=eth_ref_clk
129	<pre>set_property - dict -{ PACKAGE_PIN C16 IOSTANDARD LVCMOS33 -} [get_ports - { .phy_rst_n -}]; #IO_L20P_T3_A20_15 · Sch=eth_rstn</pre>
130	<pre>set_property -dict { PACKAGE_PIN F15 IOSTANDARD LVCMOS33 } [get_ports { phy_rx_clk }]; #IO_L14P_T2_SRCC_15-</pre>
	Sch=eth_rx_clk
131	<pre>set_property -dict { PACKAGE_PIN G16 · · IOSTANDARD LVCMOS33 } [get_ports { phy_dv }]; #IO_L13N_T2_MRCC_15 · Sch=eth_rx_dv_</pre>
132	<pre>set_property -dict { PACKAGE_PIN D18 IOSTANDARD LVCMOS33 } [get_ports { phy_rx_data[0] }]; #IO_L21N_T3_DQS_A18_15</pre>
	Sch=eth_rxd[0]
133	<pre>set_property -dict { PACKAGE_PIN_E17 IOSTANDARD LVCMOS33 } [get_ports { phy_rx_data[1] }]; #IO_L16P_T2_A28_15</pre>
134	<pre>set_property -dict { PACKAGE_PIN E18 - IOSTANDARD LVCMOS33 } [get_ports { phy_rx_data[2] }]; #IO_L21P_T3_DQS_15- Sch=eth rxd[2]</pre>
135	sch=eth_rxd[2] set propertydict-{-PACKAGE PIN-G17IOSTANDARD-LVCMOS33-}-[get ports-{-phy-rx_data[3]-}]; #IO_L18N_T2_A23_15-
122	Seterth rxd[3]
136	<pre>set property -dict { PACKAGE PIN C17 IOSTANDARD LVCMOS33 } [get ports { phy rx er }]; #IO L20N T3 A19 15 Sch=eth rxerr</pre>
137	set_property -dict { PACKAGE_PIN H16 ··· IOSTANDARD LVCMOS33 } [get_ports { phy_tx_cl }]; #IO_L13P_T2_MRCC_15
207	Scheeth tx clk
138	set property -dict { PACKAGE PIN H15 · IOSTANDARD LVCMOS33 } [get ports { phy tx en }]; #IO L19N T3 A21 VREF 15-
	Scheeth tx en
139	<pre>set property _dict { PACKAGE PIN H14 _ IOSTANDARD LVCMOS33 } [get ports { phy tx data[0] }]; #IO L15P T2 DQS 15</pre>
	Sch=eth_txd[0]
140	<pre>set_property -dict { PACKAGE_PIN_J14 IOSTANDARD LVCMOS33 } [get_ports { phy_tx_data[1] }]; #IO_L19P_T3_A22_15</pre>
	Sch=eth_txd[1]
141	<pre>set_property -dict { PACKAGE_PIN J13 - IOSTANDARD LVCMOS33 } [get_ports { phy_tx_data[2] }]; #IO_L17N_T2_A25_15-</pre>
	Sch=eth_txd[2]
142	<pre>set_property dict { PACKAGE_PIN H17 IOSTANDARD LVCMOS33 } [get_ports { phy_tx_data[3] }]; #I0_L18P_T2_A24_15</pre>
	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.