CO503: Multi-Processor System-on-Chip (MPSoC) Design

NAME: KODAGODA K.H.S.P

REG No: E/17/168

INTRODUCTION

In this practical there is an implementation of Multi-Processor System-on-Chip. In this practical there are two cpus. Between them three is a shared memory and processors share that common memory address space and communicate with each other via memory. In the part 1 there are simple producer and simple consumer. In this MPSoC there is a software implementation to procced this scenario. In the part 2 there is a dedicated hardware FIFO queue, and this can be used to increase the performance.

Part 1: Producer-Consumer Applications on a Shared Memory Multi-Processor

There are two processors they are cpu0 and cpu1. Each CPU has TIMER, JTAG UART and ON-CHIP INSTRUCTION, DATA MEMORIES. Also, both cpus are sharing ON-CHIP MEMORY. That memory is called shared memory and that is common to both cpus. Initially a new project was created, and those hardware peripherals are included to the MPSoC and 50MHz clock signal is used to drive.

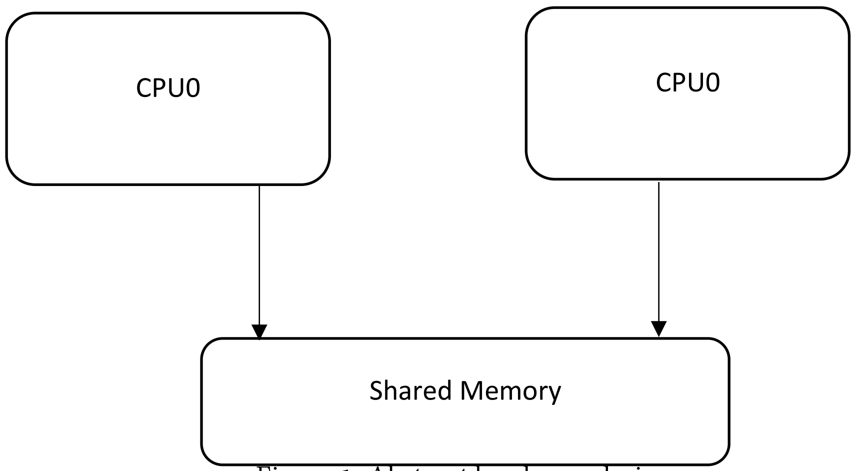


Figure 1: Abstract hardware design.

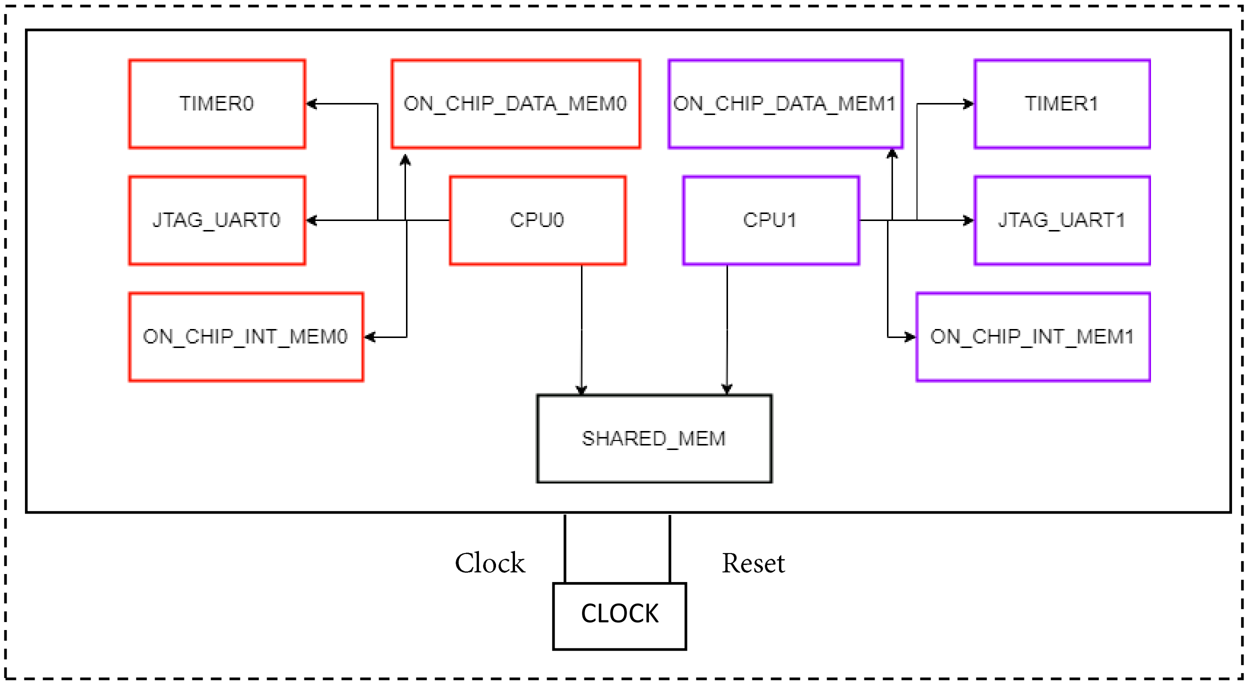
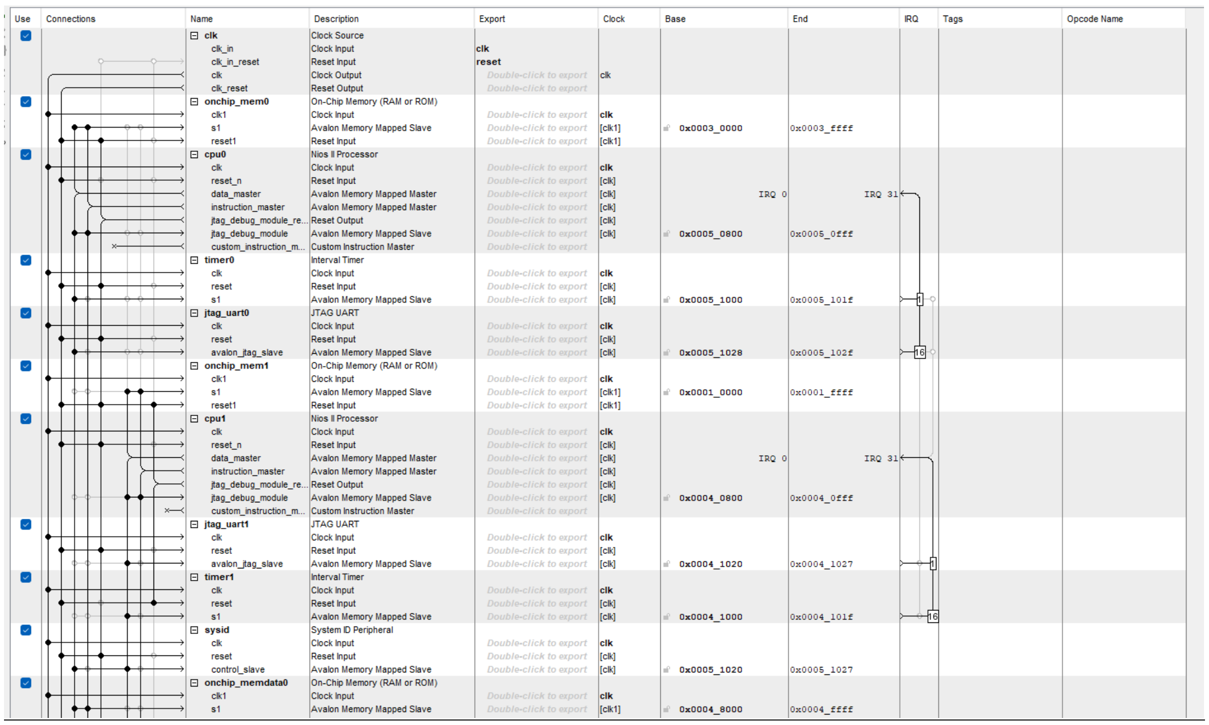


Figure 2: Hardware design of MPSoC with shared memory.

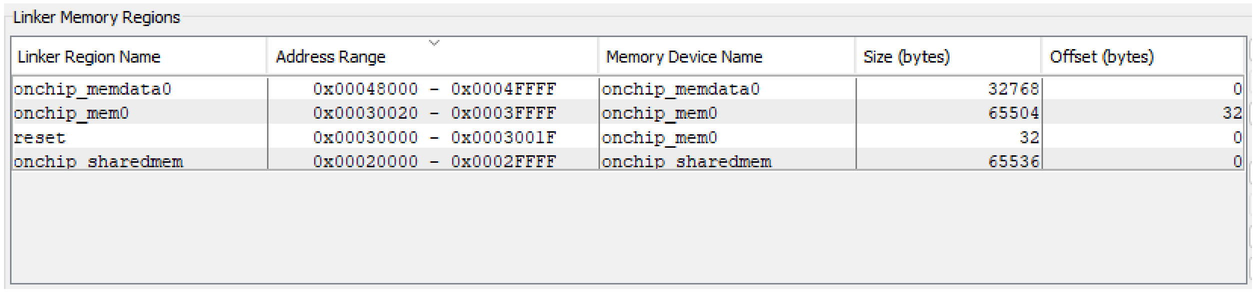


Use	Connections	Name	Description	Export	Clock	Base	End	IRQ	Tags	Opcode Name
✓		**clk**	Clock Source							
		clk_in	Clock Input	**clk**						
		clk_in_reset	Reset Input	**reset**						
		clk	Clock Output	*Double-click to export*	clk					
		clk_reset	Reset Output	*Double-click to export*						
✓		**onchip_mem0**	On-Chip Memory (RAM or ROM)							
		clk1	Clock Input	*Double-click to export*	**clk**					
		s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk1]	**0x0003_0000**	0x0003_ffff			
		reset1	Reset Input	*Double-click to export*	[clk1]					
✓		**cpu0**	Nios II Processor							
		clk	Clock Input	*Double-click to export*	**clk**					
		reset_n	Reset Input	*Double-click to export*	[clk]					
		data_master	Avalon Memory Mapped Master	*Double-click to export*	[clk]	IRQ 0	IRQ 31			
		instruction_master	Avalon Memory Mapped Master	*Double-click to export*	[clk]					
		jtag_debug_module_re...	Reset Output	*Double-click to export*	[clk]					
		jtag_debug_module	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	**0x0005_0800**	0x0005_0fff			
		custom_instruction_m...	Custom Instruction Master	*Double-click to export*						
✓		**timer0**	Interval Timer							
		clk	Clock Input	*Double-click to export*	**clk**					
		reset	Reset Input	*Double-click to export*	[clk]					
		s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	**0x0005_1000**	0x0005_101f	1		
✓		**jtag_uart0**	JTAG UART							
		clk	Clock Input	*Double-click to export*	**clk**					
		reset	Reset Input	*Double-click to export*	[clk]					
		avalon_jtag_slave	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	**0x0005_1028**	0x0005_102f	16		
✓		**onchip_mem1**	On-Chip Memory (RAM or ROM)							
		clk1	Clock Input	*Double-click to export*	**clk**					
		s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk1]	**0x0001_0000**	0x0001_ffff			
		reset1	Reset Input	*Double-click to export*	[clk1]					
✓		**cpu1**	Nios II Processor							
		clk	Clock Input	*Double-click to export*	**clk**					
		reset_n	Reset Input	*Double-click to export*	[clk]					
		data_master	Avalon Memory Mapped Master	*Double-click to export*	[clk]	IRQ 0	IRQ 31			
		instruction_master	Avalon Memory Mapped Master	*Double-click to export*	[clk]					
		jtag_debug_module_re...	Reset Output	*Double-click to export*	[clk]					
		jtag_debug_module	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	**0x0004_0800**	0x0004_0fff			
		custom_instruction_m...	Custom Instruction Master	*Double-click to export*						
✓		**jtag_uart1**	JTAG UART							
		clk	Clock Input	*Double-click to export*	**clk**					
		reset	Reset Input	*Double-click to export*	[clk]					
		avalon_jtag_slave	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	**0x0004_1020**	0x0004_1027	1		
✓		**timer1**	Interval Timer							
		clk	Clock Input	*Double-click to export*	**clk**					
		reset	Reset Input	*Double-click to export*	[clk]					
		s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	**0x0004_1000**	0x0004_101f	16		
✓		**sysid**	System ID Peripheral							
		clk	Clock Input	*Double-click to export*	**clk**					
		reset	Reset Input	*Double-click to export*	[clk]					
		control_slave	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	**0x0005_1020**	0x0005_1027			
✓		**onchip_memdata0**	On-Chip Memory (RAM or ROM)							
		clk1	Clock Input	*Double-click to export*	**clk**					
		s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk1]	**0x0004_8000**	0x0004_ffff			

Figure 3: Qsys design of MPSoC hardware.

Then two software projects are created using NIOS II software build tool. After that BSP editor is opened and then there is a tab called "Linker Script" in that window. There is section called "Linker Section Mappings" and it is used to set memory devices for sections. There are three memories for one cpu.

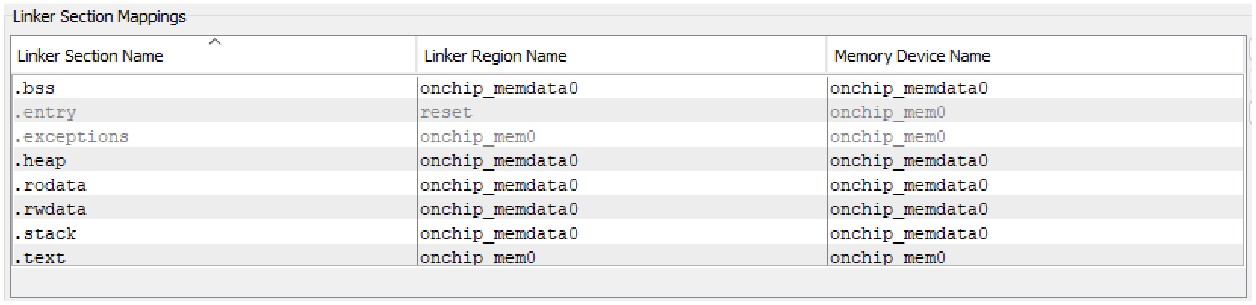
1. On-Chip data memory.
2. On-chip instruction memory.
3. Shared memory.


Linker Memory Regions

Linker Region Name	Address Range	Memory Device Name	Size (bytes)	Offset (bytes)
onchip_memdata0	0x00048000 - 0x0004FFFF	onchip_memdata0	32768	0
onchip_mem0	0x00030020 - 0x0003FFFF	onchip_mem0	65504	32
reset	0x00030000 - 0x0003001F	onchip_mem0	32	0
onchip_sharedmem	0x00020000 - 0x0002FFFF	onchip_sharedmem	65536	0

Figure 4: Memory regions

Then ".text" is set to the instruction memory and others are set to data memory.


Linker Section Mappings

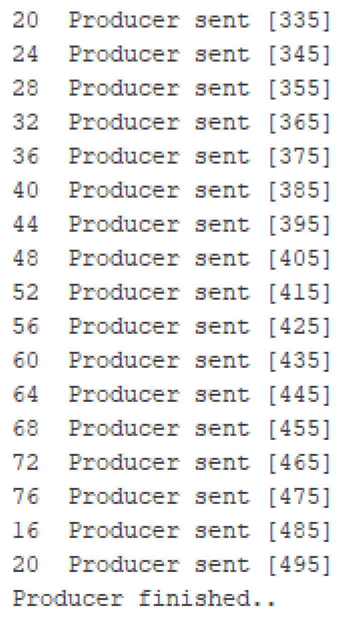
Linker Section Name	Linker Region Name	Memory Device Name
.bss	onchip_memdata0	onchip_memdata0
.entry	reset	onchip_mem0
.exceptions	onchip_mem0	onchip_mem0
.heap	onchip_memdata0	onchip_memdata0
.rodata	onchip_memdata0	onchip_memdata0
.rwdata	onchip_memdata0	onchip_memdata0
.stack	onchip_memdata0	onchip_memdata0
.text	onchip_mem0	onchip_mem0

Figure 5: Section mapping

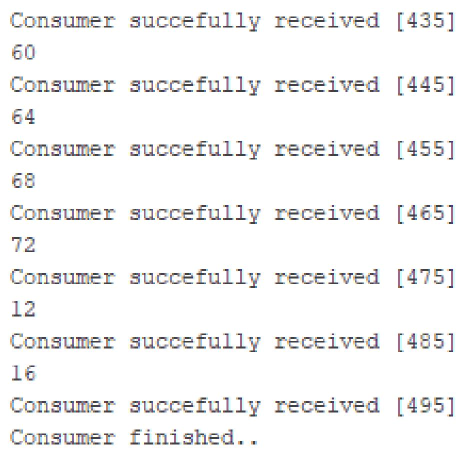
After given skeleton codes were imported and some functions are modified. There is one function for initialization and there are two other functions for writing and reading. There are three pointers, and they are writing pointer, reading pointer and full pointer. Also, there are some important variables, and one variable indicates unit size of the data block and there is another variable to store base address of memory. Then software projects were uploaded and run on both cpus.

OBSERVATIONS.

There are two cpus and one is producer and other is consumer. Producer can write data to the shared memory and consumer can read data from memory. Producer was started and it produced 16 items, The capacity of FIFO is 16 and it waited for consumer's actions. Then consumer started to read and after reading again producer wrote data to the shared memory. From 5 to 495 five by five numbers were read by consumer.



```
20  Producer sent [335]
24  Producer sent [345]
28  Producer sent [355]
32  Producer sent [365]
36  Producer sent [375]
40  Producer sent [385]
44  Producer sent [395]
48  Producer sent [405]
52  Producer sent [415]
56  Producer sent [425]
60  Producer sent [435]
64  Producer sent [445]
68  Producer sent [455]
72  Producer sent [465]
76  Producer sent [475]
16  Producer sent [485]
20  Producer sent [495]
Producer finished..
```



```
Consumer succefully received [435]
60
Consumer succefully received [445]
64
Consumer succefully received [455]
68
Consumer succefully received [465]
72
Consumer succefully received [475]
12
Consumer succefully received [485]
16
Consumer succefully received [495]
Consumer finished..
```

Figure 6: Observations of producer and consumer

PERFORMANCE.

Initially producer was started and then consumer was started. The capacity of the FIFO is 16 and it took 22 ms to write and read all items. There can be some additional latency. But somehow time difference between consumers starting and consumer finishing can be considered as the time taken to procced.

Part 2: Hardware FIFOs

This part also has same steps with some changes. In the part 1 we used software implementations and in this part, we used hardware FIFO peripheral. The on-chip FIFO memory core is a configurable component used to buffer data and provide flow control in an SOPC Builder system. This FIFI memory can be integrated easily into any SOPC Builder-generated system.

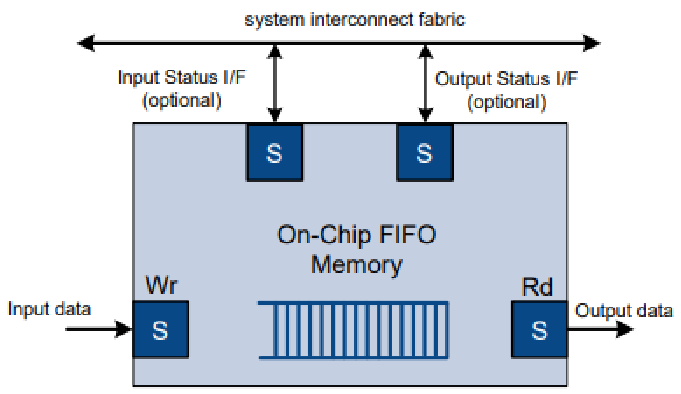
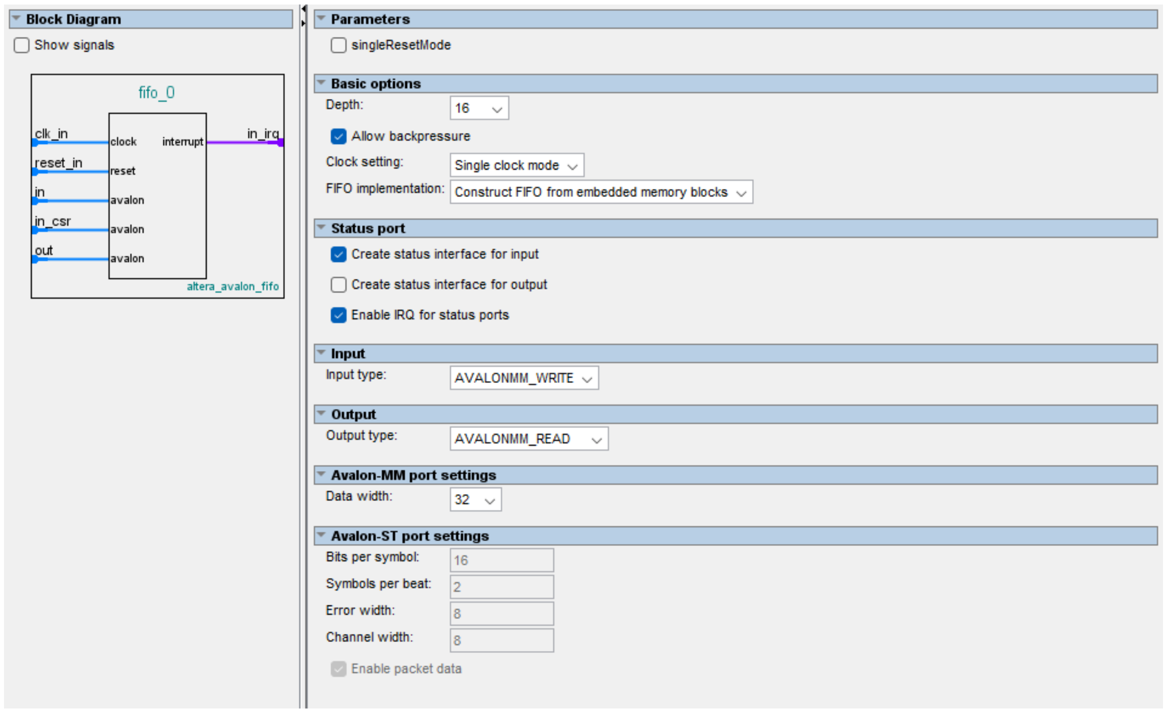

system interconnect fabric
Input Status I/F (optional)
Output Status I/F (optional)
S
S
On-Chip FIFO Memory
Wr
Rd
Input data
S
S
Output data

Figure 7: FIFO with Avalon-MM Input and Output Interfaces

This is a simple hardware and here Avalon-MM write master pushes data into the FIFO through the input interface and read master pops data through output interface. Width of the input and output are same.

There are some configurations to do in that hardware peripheral. They were configured and then software



Parameters
- [ ] singleResetMode

Basic options
- Depth: 16
- [✓] Allow backpressure
- Clock setting: Single clock mode
- FIFO implementation: Construct FIFO from embedded memory blocks

Status port
- [✓] Create status interface for input
- [ ] Create status interface for output
- [✓] Enable IRQ for status ports

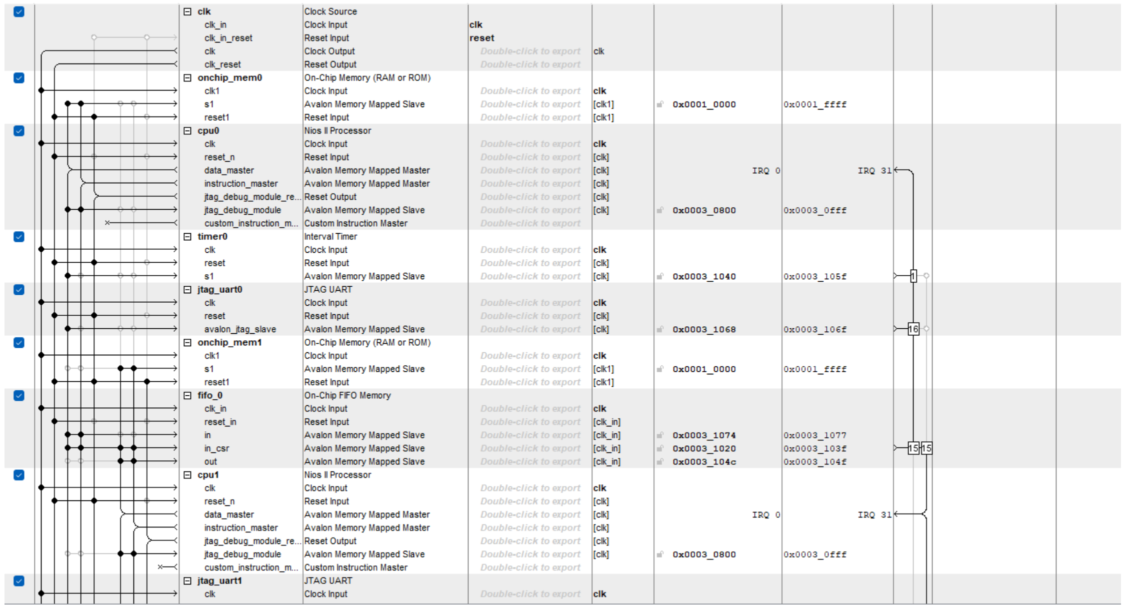
Input
- Input type: AVALONMM_WRITE

Output
- Output type: AVALONMM_READ

Avalon-MM port settings
- Data width: 32

Avalon-ST port settings
- Bits per symbol: 16
- Symbols per beat: 2
- Error width: 8
- Channel width: 8
- [✓] Enable packet data

Figure 7: Configurations of FIFO



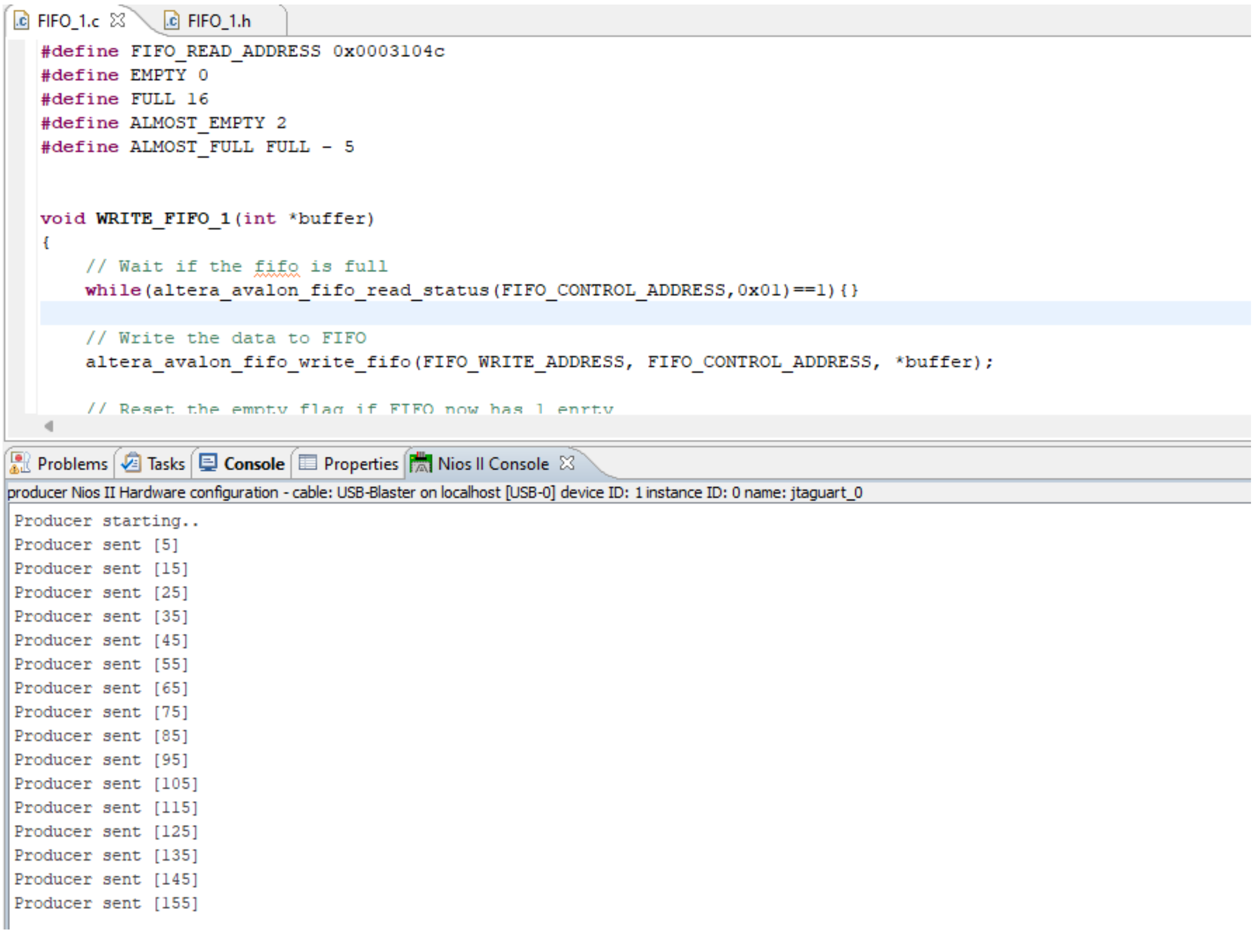
clk	Clock Source					
clk_in	Clock Input	clk				
clk_in_reset	Reset Input	reset				
clk	Clock Output	Double-click to export	clk			
clk_reset	Reset Output	Double-click to export				
onchip_mem0	On-Chip Memory (RAM or ROM)					
clk1	Clock Input	Double-click to export	clk			
s1	Avalon Memory Mapped Slave	Double-click to export	[clk1]	0x0001_0000	0x0001_ffff	
reset1	Reset Input	Double-click to export	[clk1]			
cpu0	Nios II Processor					
clk	Clock Input	Double-click to export	clk			
reset_n	Reset Input	Double-click to export	[clk]			
data_master	Avalon Memory Mapped Master	Double-click to export	[clk]	IRQ 0	IRQ 31	
instruction_master	Avalon Memory Mapped Master	Double-click to export	[clk]			
jtag_debug_module_re...	Reset Output	Double-click to export	[clk]			
jtag_debug_module	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0003_0800	0x0003_0fff	
custom_instruction_m...	Custom Instruction Master	Double-click to export				
timer0	Interval Timer					
clk	Clock Input	Double-click to export	clk			
reset	Reset Input	Double-click to export	[clk]			
s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0003_1040	0x0003_105f	1
jtag_uart0	JTAG UART					
clk	Clock Input	Double-click to export	clk			
reset	Reset Input	Double-click to export	[clk]			
avalon_jtag_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0003_1068	0x0003_106f	16
onchip_mem1	On-Chip Memory (RAM or ROM)					
clk1	Clock Input	Double-click to export	clk			
s1	Avalon Memory Mapped Slave	Double-click to export	[clk1]	0x0001_0000	0x0001_ffff	
reset1	Reset Input	Double-click to export	[clk1]			
fifo_0	On-Chip FIFO Memory					
clk_in	Clock Input	Double-click to export	clk			
reset_in	Reset Input	Double-click to export	[clk_in]			
in	Avalon Memory Mapped Slave	Double-click to export	[clk_in]	0x0003_1074	0x0003_1077	
in_csr	Avalon Memory Mapped Slave	Double-click to export	[clk_in]	0x0003_1020	0x0003_103f	15 15
out	Avalon Memory Mapped Slave	Double-click to export	[clk_in]	0x0003_104c	0x0003_104f	
cpu1	Nios II Processor					
clk	Clock Input	Double-click to export	clk			
reset_n	Reset Input	Double-click to export	[clk]			
data_master	Avalon Memory Mapped Master	Double-click to export	[clk]	IRQ 0	IRQ 31	
instruction_master	Avalon Memory Mapped Master	Double-click to export	[clk]			
jtag_debug_module_re...	Reset Output	Double-click to export	[clk]			
jtag_debug_module	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0003_0800	0x0003_0fff	
custom_instruction_m...	Custom Instruction Master	Double-click to export				
jtag_uart1	JTAG UART					
clk	Clock Input	Double-click to export	clk			

Figure 8: Qsys design of MPSoC hardware with FIFO

After generating the hardware design, it was uploaded to the FPGA board. Also, there are two software projects for producer and consumer. There are three functions to complete in the software project and they were coded according to the manual of FIFO. One function was used for initialization and other two functions were used for writing and reading. There are in-built functions and they could be used to that software implementation.

OBSERVATIONS.

When producer was started then it produced 16 items to the FIFO and it waited for reading of the consumer. The depth of the FIFO and once it could produce 16 items only. The consumer started to read the data and after reading producer produced rest of the items to the FIFO queue.



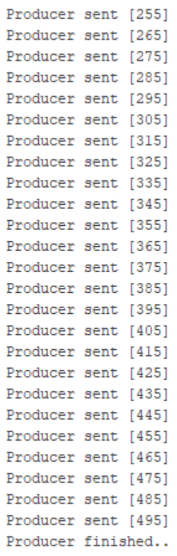
```
#define FIFO_READ_ADDRESS 0x0003104c
#define EMPTY 0
#define FULL 16
#define ALMOST_EMPTY 2
#define ALMOST_FULL FULL - 5


void WRITE_FIFO_1(int *buffer)
{
    // Wait if the fifo is full
    while(altera_avalon_fifo_read_status(FIFO_CONTROL_ADDRESS,0x01)==1){}

    // Write the data to FIFO
    altera_avalon_fifo_write_fifo(FIFO_WRITE_ADDRESS, FIFO_CONTROL_ADDRESS, *buffer);

    // Reset the empty flag if FIFO now has 1 enrty
```

producer Nios II Hardware configuration - cable: USB-Blaster on localhost [USB-0] device ID: 1 instance ID: 0 name: jtaguart_0

```
Producer starting..
Producer sent [5]
Producer sent [15]
Producer sent [25]
Producer sent [35]
Producer sent [45]
Producer sent [55]
Producer sent [65]
Producer sent [75]
Producer sent [85]
Producer sent [95]
Producer sent [105]
Producer sent [115]
Producer sent [125]
Producer sent [135]
Producer sent [145]
Producer sent [155]
```

Figure 9: Producer waiting for consumers consuming

Finally producer produced all data items and items were consumed by consumer.



```
Producer sent [255]
Producer sent [265]
Producer sent [275]
Producer sent [285]
Producer sent [295]
Producer sent [305]
Producer sent [315]
Producer sent [325]
Producer sent [335]
Producer sent [345]
Producer sent [355]
Producer sent [365]
Producer sent [375]
Producer sent [385]
Producer sent [395]
Producer sent [405]
Producer sent [415]
Producer sent [425]
Producer sent [435]
Producer sent [445]
Producer sent [455]
Producer sent [465]
Producer sent [475]
Producer sent [485]
Producer sent [495]
Producer finished..
```

Figure 10: Completion of the process.

DEFFICULTIES.

Initially IRQ values for FIFO was not set. Then it produced a error and the set IRQ values of FIFO.

Initialization of software implementation was done twice in the producer and consumer. Because of that after consumer again updated the pointer values during the process. So it caused to behavior of the FIFO and then initialization in the consumer software implementation was cleared.

CONCLUSION.

Shared memory implementation is a method which can be used to communicate between two processors in the multiprocessor system. To complete the process software implementation needed 22ms and to complete the process hardware implementation only needed 16ms. So hardware implementation of FIFO is better than software implementation of FIFO according to the performance. Also, accuracy of the hardware implementation is high.