**Documentation of SHA1 IP**

The SHA1 IP communicates via AXI-lite interface with 32 bits data bus. It allows only unidirectional communication as of now.



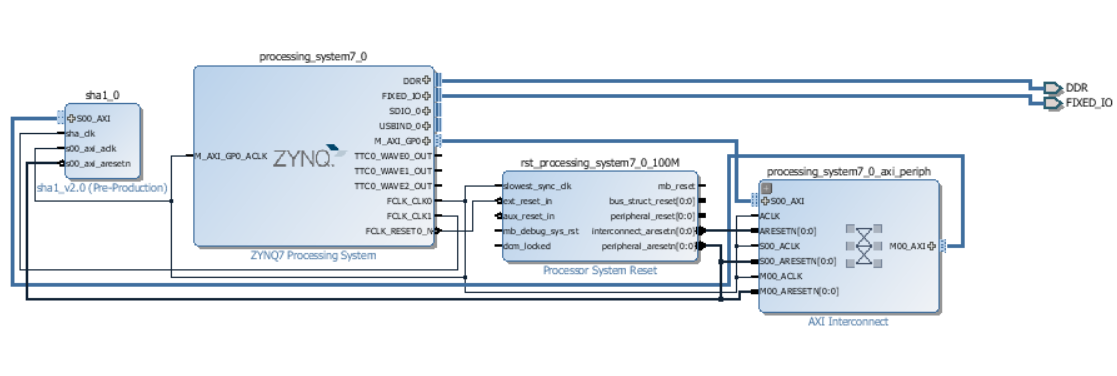
**The addresses of registers are as follows:**

* ADDR\_CTRL = 8’h08
* CTRL\_INIT\_BIT = 0
* CTRL\_NEXT\_BIT = 1
* ADDR\_STATUS = 8’h09
* STATUS\_READY\_BIT = 0
* STATUS\_VALID\_BIT = 1
* ADDR\_BLOCK0 = 8’h10
* ADDR\_BLOCK1 = 8’h11
* ADDR\_BLOCK2 = 8’h12
* ADDR\_BLOCK3 = 8’h13
* ADDR\_BLOCK4 = 8’h14
* ADDR\_BLOCK5 = 8’h15
* ADDR\_BLOCK6 = 8’h16
* ADDR\_BLOCK7 = 8’h17
* ADDR\_BLOCK8 = 8’h18
* ADDR\_BLOCK9 = 8’h19
* ADDR\_BLOCK10 = 8’h1a
* ADDR\_BLOCK11 = 8’h1b
* ADDR\_BLOCK12 = 8’h1c
* ADDR\_BLOCK13 = 8’h1d
* ADDR\_BLOCK14 = 8’h1e
* ADDR\_BLOCK15 = 8’h1f
* ADDR\_DIGEST0 = 8’h20
* ADDR\_DIGEST1 = 8’h21
* ADDR\_DIGEST2 = 8’h22
* ADDR\_DIGEST3 = 8’h23
* ADDR\_DIGEST4 = 8’h24
* CORE\_NAME0 = 32’h73686131 // "sha1"
* CORE\_NAME1 = 32’h20202020 // " "
* CORE\_VERSION = 32’h302e3530 // "0.50"
* CTRL\_INIT\_VALUE = 8’h01
* CTRL\_NEXT\_VALUE = 8’h02

The ip has been tested with core-clk of 100Mhz and axi-clk of 100Mhz.

**Note:** As of now, both core and axi clocks should be same. Future updates may allow using different clocks

**Example design:**



**Test Code:**

**#include** <stdio.h>

**#include** "platform.h"

**#include** "xil\_printf.h"

**#include** "address.h"

**#include** "xil\_io.h"

**#include** "sleep.h"

**int** **main**()

{

**int** ready,i,out\_temp;

init\_platform();

print("Hello World\n\r");

Xil\_Out32(XPAR\_SHA1\_0\_S00\_AXI\_BASEADDR+(ADDR\_BLOCK0<<2), 0X61626380);

Xil\_Out32(XPAR\_SHA1\_0\_S00\_AXI\_BASEADDR+(ADDR\_BLOCK1<<2), 0X00000000);

Xil\_Out32(XPAR\_SHA1\_0\_S00\_AXI\_BASEADDR+(ADDR\_BLOCK2<<2), 0X00000000);

Xil\_Out32(XPAR\_SHA1\_0\_S00\_AXI\_BASEADDR+(ADDR\_BLOCK3<<2), 0X00000000);

Xil\_Out32(XPAR\_SHA1\_0\_S00\_AXI\_BASEADDR+(ADDR\_BLOCK4<<2), 0X00000000);

Xil\_Out32(XPAR\_SHA1\_0\_S00\_AXI\_BASEADDR+(ADDR\_BLOCK5<<2), 0X00000000);

Xil\_Out32(XPAR\_SHA1\_0\_S00\_AXI\_BASEADDR+(ADDR\_BLOCK6<<2), 0X00000000);

Xil\_Out32(XPAR\_SHA1\_0\_S00\_AXI\_BASEADDR+(ADDR\_BLOCK7<<2), 0X00000000);

Xil\_Out32(XPAR\_SHA1\_0\_S00\_AXI\_BASEADDR+(ADDR\_BLOCK8<<2), 0X00000000);

Xil\_Out32(XPAR\_SHA1\_0\_S00\_AXI\_BASEADDR+(ADDR\_BLOCK9<<2), 0X00000000);

Xil\_Out32(XPAR\_SHA1\_0\_S00\_AXI\_BASEADDR+(ADDR\_BLOCK10<<2), 0X00000000);

Xil\_Out32(XPAR\_SHA1\_0\_S00\_AXI\_BASEADDR+(ADDR\_BLOCK11<<2), 0X00000000);

Xil\_Out32(XPAR\_SHA1\_0\_S00\_AXI\_BASEADDR+(ADDR\_BLOCK12<<2), 0X00000000);

Xil\_Out32(XPAR\_SHA1\_0\_S00\_AXI\_BASEADDR+(ADDR\_BLOCK13<<2), 0X00000000);

Xil\_Out32(XPAR\_SHA1\_0\_S00\_AXI\_BASEADDR+(ADDR\_BLOCK14<<2), 0X00000000);

Xil\_Out32(XPAR\_SHA1\_0\_S00\_AXI\_BASEADDR+(ADDR\_BLOCK15<<2), 0X00000018);

Xil\_Out32(XPAR\_SHA1\_0\_S00\_AXI\_BASEADDR+(ADDR\_CTRL<<2), CTRL\_INIT\_VALUE);

usleep(10);

Xil\_Out32(XPAR\_SHA1\_0\_S00\_AXI\_BASEADDR+(ADDR\_CTRL<<2), 0X00);

**do**{

ready = (Xil\_In32(XPAR\_SHA1\_0\_S00\_AXI\_BASEADDR+(ADDR\_STATUS<<2))) & 0X00000001;

}**while**(!ready);

**for**(i=0;i<5;i++){

out\_temp = Xil\_In32(XPAR\_SHA1\_0\_S00\_AXI\_BASEADDR+((ADDR\_DIGEST0 + i)<<2));

xil\_printf("%08x",out\_temp);

}

print("\n");

**for**(i=0;i<5;i++){

out\_temp = Xil\_In32(XPAR\_SHA1\_0\_S00\_AXI\_BASEADDR+((ADDR\_DIGEST0 + i)<<2));

xil\_printf("%08x",out\_temp);

}

print("\n");

cleanup\_platform();

**return** 0;

}

**Result of Testcode:**

A9993E364706816ABA3E25717850C26C9CD0D89D

**Note:** When encrypting next block without initializing again, use CTRL\_REG value as 2