ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 6, Issue 5, November 2016

Programmable Pulse Doppler radar Simulator

Saad Daoud AL-Shamma
Electronic Department, College of Engineering, Ninevah University

Abstract— In this paper a programmable pulse Doppler radar simulator has been designed and implemented based on system on chip technology (SOC) .all the hardware has been implemented inside the programmable logic (PL) and the driven software has been stored and executed by processor system (PS) of the chip. AXI bus is chosen for data transfer between PS and PL portions of the chip. HLS based on System C language which is new classes added to the C++ language for hardware design is used . A test benches have been written for the simulation, checking and the good functionality of the system.

Index Terms— **SOC, System C, HLS, AXI.**

I. INTRODUCTION

A System C is a system design language that has evolved in response to a pervasive need for a language that improves overall productivity for designers of electronic systems. Typically, today's systems contain application-specific hardware and software. Furthermore, the hardware and software are usually co-developed on a tight schedule, the systems have tight real-time performance constraints, and thorough functional verification is required to avoid expensive and sometimes catastrophic failures.

System C offers real productivity gains by letting engineers design both the hardware and software components together as these components would exist on the final system, but at a high level of abstraction. This higher level of abstraction gives the design team a fundamental understanding early in the design process of the intricacies and interactions of the entire system and enables better system tradeoffs, better and earlier verification, and overall productivity gains through reuse of early system models as executable specifications

Strictly speaking, SystemC is not a language, but rather a class library within a well-established language, C++. SystemC is not a panacea that will solve every design productivity issue. However, when SystemC is coupled

With the SystemC Verification Library, it does provide in one language many of the characteristics relevant to system design and modeling tasks that are missing or scattered among the other languages. Additionally, SystemC provides a common language for software and hardware, C++.

Several languages have emerged to address the various aspects of system design. Although Ada and Java have proven their value, C/C++ is predominately used today for embedded system software. The hardware description languages (HDLs), VHDL and Verilog, are used for simulating and synthesizing digital circuits. Vera and e are the languages of choice for Functional verification of complex application-specific integrated circuits (ASICs). System Verilog is a new language that evolves the Verilog language to address many hardware-oriented system design issues. Matlab and several other tools and languages such as SPW and System Studio are widely used for capturing system requirements and developing signal processing algorithms. Fig. 1 highlights the application of these and other system design languages [1]

A. For pulse Doppler radar simulator the main signals needed is [2]

1- SYNC (Trigger)which represent the start of transmission and ON time for the transmitter
2- Range Gate (RG) which represent the minimum resolution in range for the radar And equal to SYNC ON time
3- ACP represent the minimum resolution in azimuth (4096 pulses/revolution)
4- NP north pulse (one pulse /revolution)
5- Video one or two channels which represent the output of phase detector

II. SIGNALS GENERATION

A SOC development board Zed Board using the Xilinx Zynq®-7000 All Programmable SoC. [3] with VIVADO HLS 2015.4 development tool from Xilinx are used for development ,simulation and testing [4]

A. SYNC and RG

A counter with the system clock is used .the counter is reset when the out is greater or equal to a programmed value from PS (sr_t_rg) and range gate clock (sr_rg) change state .

Another counter with rg as clock is used for the number of range gates in one SYNC is reset when the counter is greater or equal to programmed value from PS (sr_t_tr) The above counters are described using SystemC [4] language as shown in Fig. 2

B. ACP and NP

A counter with the system clock is used .the counter is reset when the out is greater or equal to a programmed value from PS (sr_t_acp) and azimuth gate clock (sr_acp) change state .

Another counter with sr_acp as clock is used for azimuth clocks in one revolution is reset when the counter is greater or equal 4096 as shown in Fig. 3

C. Video generation

Dual port Memory RAM (4K*8) is used to store the video for a complete SYNC pulse .The write operation is controlled by the PS while the reading is done by range gate counter .

The write operation is enabled when sr_we = true and read when sr_we = false. As shown in Fig. 3

D. Processor (PS) and logic connection (PL)

Using Export RTL tool in Vivado HLS an intellectual property (ip) ,(sradar_0) is generated and added to design repository with AXI slave (S_AXI_SLV0)connection to the PS as defined by Protocol directive as shown in Fig. 4, 5

III. ANALYSIS AND RESULTS

Following is the Vivado HLS design flow: [5]
1. Compile, execute (simulate), and debug the C algorithm.
Note: In high-level synthesis, running the compiled C program is referred to as *C simulation*.
Executing the C algorithm simulates the function to validate that the algorithm is functionally correct.
2. Synthesize the C algorithm into an RTL implementation, optionally using user optimization directives.
3. Generate comprehensive reports and analyze the design.
4. Verify the RTL implementation using a pushbutton flow.
5. Package the RTL implementation into a selection of IP formats
- The synthesis process reports indicate the following parameters.
The minimum clock for the design 4.96 ns (3.71 ns + 1.25 ns),The maximum clock number for one iteration in the loop is 6 and The percentages of resources used are less than 1 % As shown in Fig. 6
- To find the relation between the number of clocks (sr_t_rg) which is an input from the PS through AXI bus and range gate width generated by PL side .A different simulations ,synthesizes and C/Cosimulation was done with different values for (sr_t_rg) and the following results are obtained as show in Fig. 7
3- From Fig. 7 the relation between sr_t_rg and Range Gate Width is
Range Gate Width = rs_t_rg *80 + 80
Ex- for sr_t_rg = 50
Range Gate Width = 50 * 80 + 80 =4080 ns
The same analysis for ACP (sr_t_acp) which defines the number of revolution per minute for the radar.
Fig. 8 shows the simulation for
sr_t_rg = 50 (define range gate –rg)
sr_t_acp = 51 (define ACP)
sr_t_tr = 6 (define number of range gates /2 in one Trigger)
sr_t_np = 10 (define number of ACP/2 in one revolution)
And memory testing by writing 23 in the addresses 0 to 100 with sr_we = 1 and reading with sr_we = 0
4- To test the ip generated and the connection with the PS a C- Program is written to write 23 value in memory location 20 and continuous reading of azimuth and video output and display on PC connected as terminal as shown in Fig. 9

IV. CONCLUSIONS

1- Different types of radar signals can be easily generated by writing a suitable C- Program software and setting new parameters for radar signals to be loaded in the PS side
2- The use of HLS design tools will reduce dramatically the design time and easily generate intellectual properties(ip) which could reused and integrated for other designs

REFERENCES

[1] David C. Black and Jack Donovan “SystemC from the Ground up” Eklectic Ally, Inc. 2004 .pp. 1-3 Chapter 1.

[2] Mahmod, A. Al_Zubaidy, Saad S. Al_Samaa, Khalil H. Sayidmarie, ”A PC-BASED RADAR SYSTEM SIMULATOR”, IEEE 2005 International Symposium on Microwave, Antenna, Propagation and EMC Technologies For Wireless Communications (MAPE 2005), 8- 12 Aug., 2005, Beijing, China.

[3] www.zedboard.org.

[4] SYSTEMC version 2.0 user guide (http://www.systemc.org).

[5] Vivado Design Suite User Guide High-Level Synthesis Xilinx UG902 (V2016.2 June 8, 2016).

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 6, Issue 5, November 2016

APPENDIX

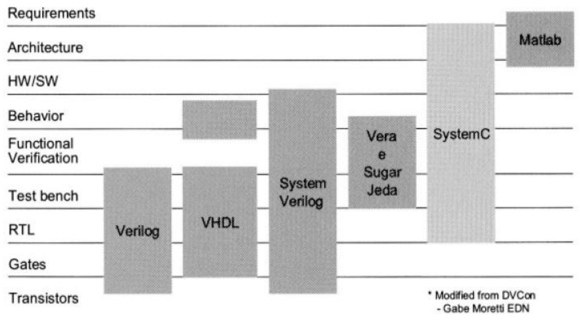
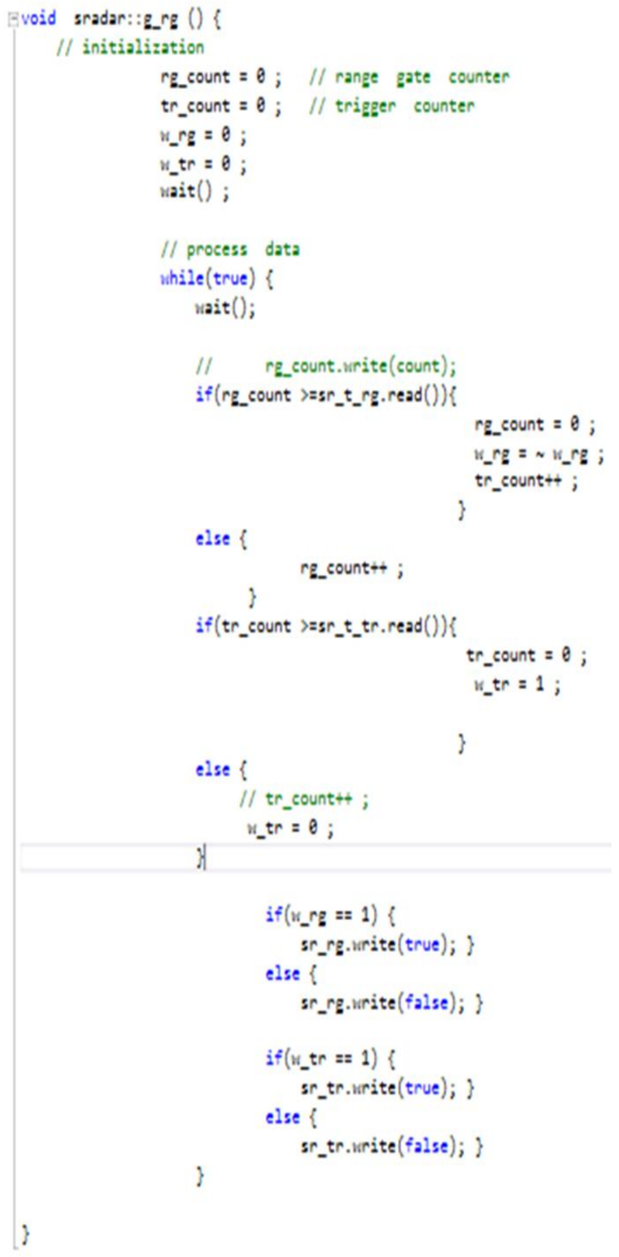
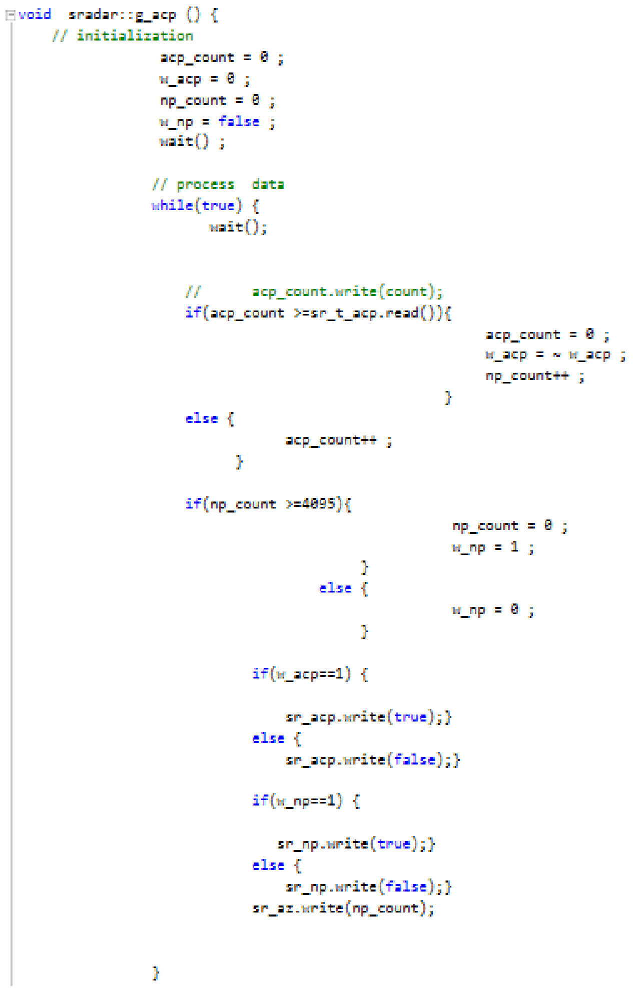


Fig. 1(SystemC constructed with other designs languages) [1]



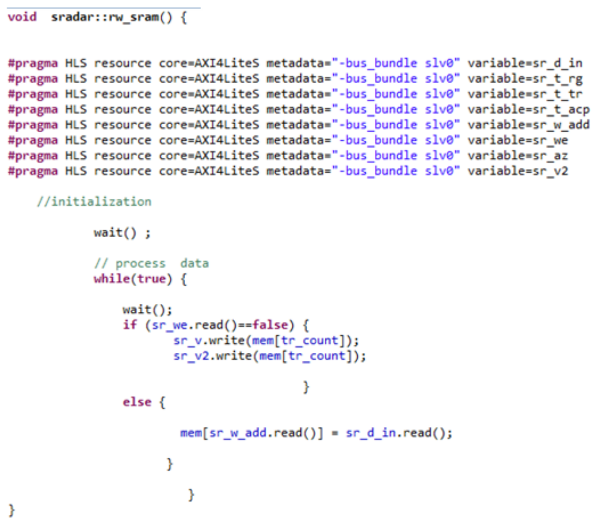
```
void  sradar::g_rg () {
    // initialization
                rg_count = 0 ;   // range  gate  counter
                tr_count = 0 ;   // trigger  counter
                w_rg = 0 ;
                w_tr = 0 ;
                wait() ;

                // process  data
                while(true) {
                    wait();

                    //      rg_count.write(count);
                    if(rg_count >=sr_t_rg.read()){
                                        rg_count = 0 ;
                                        w_rg = ~ w_rg ;
                                        tr_count++ ;
                                        }
                    else {
                                rg_count++ ;
                            }
                    if(tr_count >=sr_t_tr.read()){
                                        tr_count = 0 ;
                                        w_tr = 1 ;

                                        }
                    else {
                        // tr_count++ ;
                            w_tr = 0 ;
                    }

                        if(w_rg == 1) {
                            sr_rg.write(true); }
                        else {
                            sr_rg.write(false); }

                        if(w_tr == 1) {
                            sr_tr.write(true); }
                        else {
                            sr_tr.write(false); }
                    }

}
```

Fig. 2(RG and SYNC generation)



```
void  sradar::g_acp () {
    // initialization
                acp_count = 0 ;
                w_acp = 0 ;
                np_count = 0 ;
                w_np = false ;
                wait() ;

               // process  data
               while(true) {
                    wait();


                  //      acp_count.write(count);
                  if(acp_count >=sr_t_acp.read()){
                                                 acp_count = 0 ;
                                                 w_acp = ~ w_acp ;
                                                 np_count++ ;
                                             }
                  else {
                              acp_count++ ;
                        }

                  if(np_count >=4095){
                                              np_count = 0 ;
                                              w_np = 1 ;
                                   }
                               else {
                                              w_np = 0 ;
                                   }

                       if(w_acp==1) {

                          sr_acp.write(true);}
                       else {
                          sr_acp.write(false);}

                       if(w_np==1) {

                         sr_np.write(true);}
                       else {
                          sr_np.write(false);}
                       sr_az.write(np_count);


               }
```

Fig. 3(ACP and NP generation)



```
void  sradar::rw_sram() {


#pragma HLS resource core=AXI4LiteS metadata="-bus_bundle slv0" variable=sr_d_in
#pragma HLS resource core=AXI4LiteS metadata="-bus_bundle slv0" variable=sr_t_rg
#pragma HLS resource core=AXI4LiteS metadata="-bus_bundle slv0" variable=sr_t_tr
#pragma HLS resource core=AXI4LiteS metadata="-bus_bundle slv0" variable=sr_t_acp
#pragma HLS resource core=AXI4LiteS metadata="-bus_bundle slv0" variable=sr_w_add
#pragma HLS resource core=AXI4LiteS metadata="-bus_bundle slv0" variable=sr_we
#pragma HLS resource core=AXI4LiteS metadata="-bus_bundle slv0" variable=sr_az
#pragma HLS resource core=AXI4LiteS metadata="-bus_bundle slv0" variable=sr_v2

    //initialization

            wait() ;

            // process  data
            while(true) {

                wait();
                if (sr_we.read()==false) {
                       sr_v.write(mem[tr_count]);
                       sr_v2.write(mem[tr_count]);

                                   }
                else {

                        mem[sr_w_add.read()] = sr_d_in.read();

                      }

                         }
}
```

Fig. 4(Memory read write and protocols directives)

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 6, Issue 5, November 2016

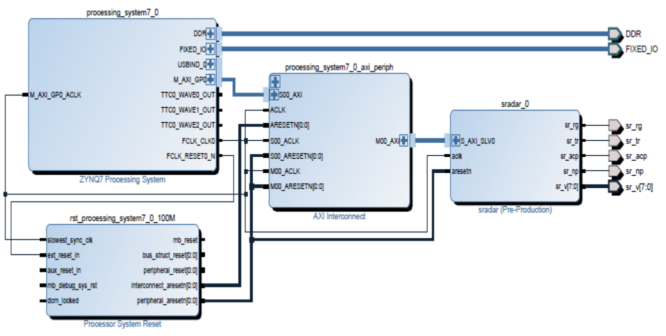


Fig. 5(PS and ip connection)

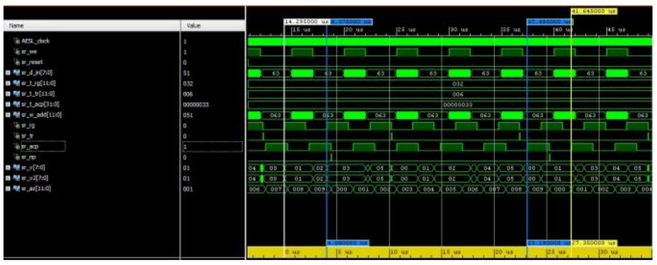
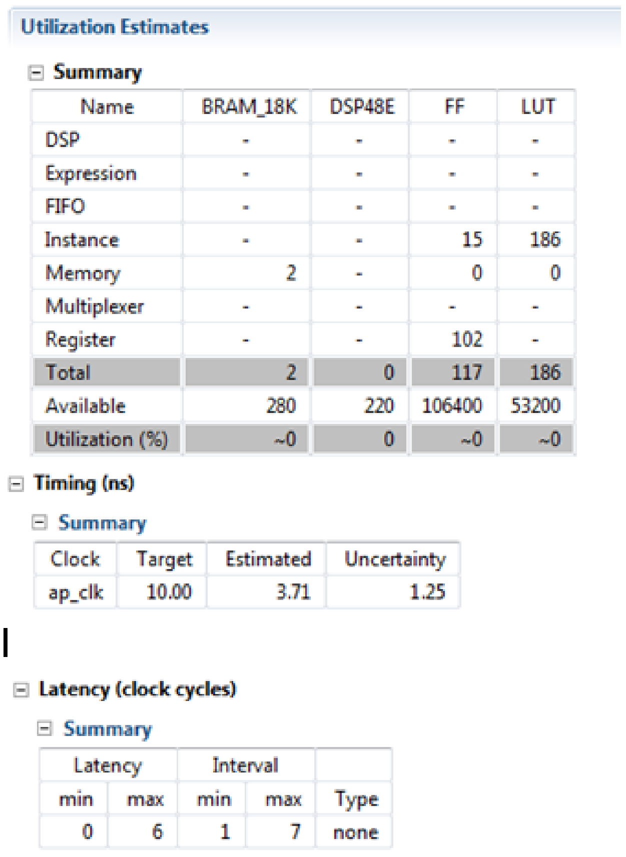


Fig. 8(simulation results)



Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	-	-
FIFO	-	-	-	-
Instance	-	-	15	186
Memory	2	-	0	0
Multiplexer	-	-	-	-
Register	-	-	102	-
Total	2	0	117	186
Available	280	220	106400	53200
Utilization (%)	~0	0	~0	~0

Timing (ns)

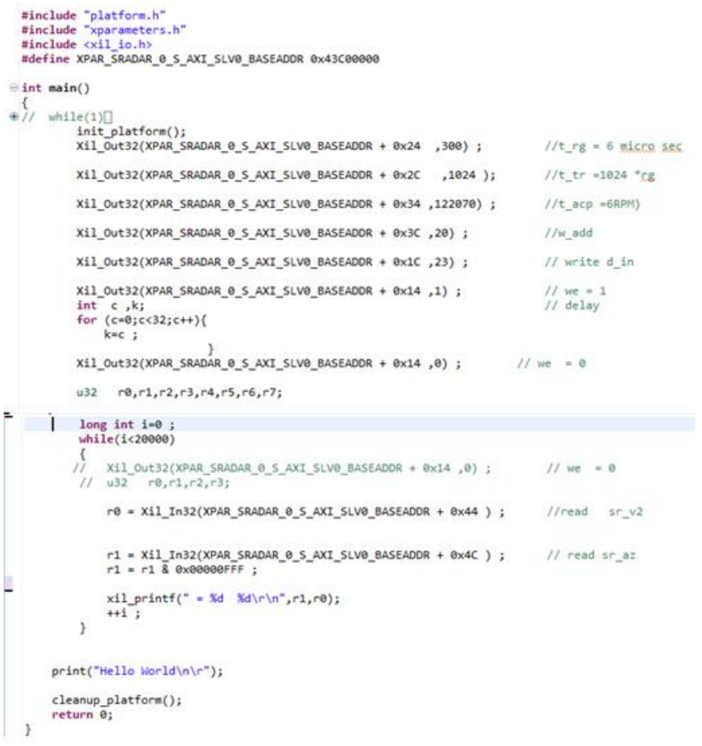
Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	3.71	1.25

Latency (clock cycles)

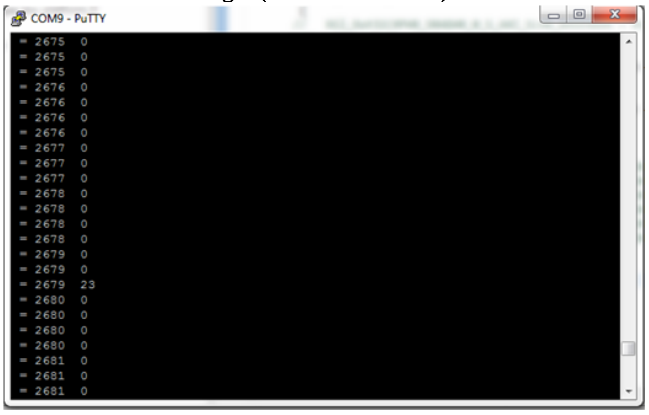
Summary

Latency		Interval		
min	max	min	max	Type
0	6	1	7	none



```
#include "platform.h"
#include "xparameters.h"
#include <xil_io.h>
#define XPAR_SRADAR_0_S_AXI_SLV0_BASEADDR 0x43C00000

int main()
{
//  while(1)
        init_platform();
        Xil_Out32(XPAR_SRADAR_0_S_AXI_SLV0_BASEADDR + 0x24  ,300) ;          //t_rg = 6 micro sec

        Xil_Out32(XPAR_SRADAR_0_S_AXI_SLV0_BASEADDR + 0x2C   ,1024 );        //t_tr =1024 *rg

        Xil_Out32(XPAR_SRADAR_0_S_AXI_SLV0_BASEADDR + 0x34 ,122070) ;        //t_acp =6RPM)

        Xil_Out32(XPAR_SRADAR_0_S_AXI_SLV0_BASEADDR + 0x3C ,20) ;            //w_add

        Xil_Out32(XPAR_SRADAR_0_S_AXI_SLV0_BASEADDR + 0x1C ,23) ;            // write d_in

        Xil_Out32(XPAR_SRADAR_0_S_AXI_SLV0_BASEADDR + 0x14 ,1) ;             // we = 1
        int  c ,k;                                                            // delay
        for (c=0;c<32;c++){
            k=c ;
                          }
        Xil_Out32(XPAR_SRADAR_0_S_AXI_SLV0_BASEADDR + 0x14 ,0) ;          // we  = 0

        u32   r0,r1,r2,r3,r4,r5,r6,r7;

        long int i=0 ;
        while(i<20000)
        {
      //    Xil_Out32(XPAR_SRADAR_0_S_AXI_SLV0_BASEADDR + 0x14 ,0) ;          // we  = 0
       //   u32   r0,r1,r2,r3;

            r0 = Xil_In32(XPAR_SRADAR_0_S_AXI_SLV0_BASEADDR + 0x44 ) ;      //read   sr_v2


            r1 = Xil_In32(XPAR_SRADAR_0_S_AXI_SLV0_BASEADDR + 0x4C ) ;      // read sr_az
            r1 = r1 & 0x00000FFF ;

            xil_printf(" = %d  %d\r\n",r1,r0);
            ++i ;
        }


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

    cleanup_platform();
    return 0;
}
```

Fig. 9(PS test Software)



```
= 2675  0
= 2675  0
= 2675  0
= 2676  0
= 2676  0
= 2676  0
= 2676  0
= 2677  0
= 2677  0
= 2677  0
= 2678  0
= 2678  0
= 2678  0
= 2678  0
= 2679  0
= 2679  0
= 2679  23
= 2680  0
= 2680  0
= 2680  0
= 2681  0
= 2681  0
= 2681  0
```

Fig. 10(reading azimuth counter and video out)

Sr_t_rg	Rang gate width (ns)
1	160
2	240
3	320
4	400
22	1860
23	1920
50	4080
51	4160

Fig. 7(RG time parameters and RG width)

Fig. 6(syntheses results)