Military Technical College
Kobry Elkobbah,
Cairo, Egypt

5th International Conference
on Electrical Engineering
ICEENG 2006

An Implementation of the Run-Length Decode Algorithm using FPGA

Gouda Ismail Salama*,Ph.D. Fawzy ELtothamy Hassan*,Ph.D.

Ramy Mohammed Bahy*,M.Sc., Sameh shawky Ibrahim*, B.Sc.

Abstract:

This paper presents a real time implementation of Run-Length Decode (RLD) using FPGA as one of image decompression algorithms. The RLD algorithm is the decoder of the Run-Length Encode. RLD can be implemented either on commercial DSP or as an ASIC but due to the huge development in the FPGA field, it is recommended to use the FPGA technology. The design steps from design entry to files which are needed for the download process are developed. Also, the method of testing the downloaded design is explained.

Keyword: Run-Length Decode, FPGA, Image compression

1- Introduction:

Image compression process is the reduction of the number of bits required to represent a digital image [1], Run-Length Encode (RLE) is considered one of a simple image compression algorithms which depends on the reduction of the repeated image gray levels values into only two values, the first value represents the gray level value and the second one represents the number of repetition of this gray level value [2]. The Run-Length Decode (RLD) is the inverse of the RLE. The RLD depends on the reconstruction of the original image gray levels from the compressed image (gray levels values and their number of repetitions). The RLD is required after RLE in order to reconstruct the image after the compression process; this operation is called the decompression process. In previous work an implementation of RLE using FPGA was developed [3]. In this paper an implementation of RLD using FPGA is introduced as a continuation of the previous work. The speed is one of the main factors in evaluation of a compression and decompression algorithms. In some applications such as On-Board satellite image compression, the speed of the compression / decompression algorithms is mandatory due to real time nature of the satellite missions (simultaneous imaging and downlinking). Therefore, hardware-based solutions are considered [3-4-5]. The design steps will be accomplished by using two well known packages: **FPGA advantage for HDL design**, **release 5.2** and **Xilinx ISE, release 5.2i**.
In this paper**,** all design steps which includes design entry, functional simulation, synthesis and timing simulation [6] are introduced. The RLD design description is introduced in the next section, the simulation results are explained in section 3 , testing the RLD downloaded design is explained in section 4 and finally conclusion is presented in the last section.

* Egyptian Armed Forces

2- RLD design description

Fig.1. illustrates the main inputs and outputs to the proposed design. The function of inputs/outputs signals are given in Table 1.

Table 1. Main Block inputs/outputs

Port	Function
input_Data	Enters the gray levels values to RAM 3.
input_Rep	Enters the number of repetition of the gray levels values to RAM 4.
iena	Enables the memories (RAM 3 and RAM 4) in order to be in writing state.
oena	Enables the memories (RAM 3 and RAM 4) in order to be in reading state
addr_in	Enters the values of addresses to RAM 3 and RAM 4.
Decomp_data	The output decompressed data.
rst	**Reset,** when rst='**1**' the circuit is in reset state and no processing occurs, when rst='**0**' the circuit is ready for processing procedures.
clk	**Clock,** at every clock rising edge a new compressed image data is entered
Ena	**Enable** of the circuit (Ena='**1**' the circuit is 'ON' and ready for processing steps, Ena='**0**' the cicuit is 'OFF' and no processing sequence occurs).
count	Indicates the number of the output decompressed data.

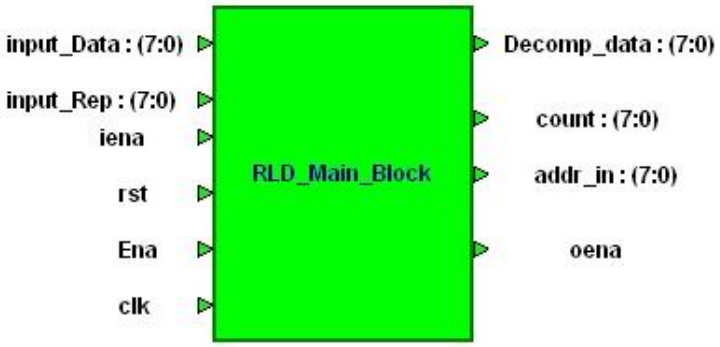

Fig.1. The Main Block of the RLD Algorithm

The compressed image data (gray levels values and its repetitions values) is stored in two memories then the RLD processing is performed on this data. In Fig. 2, the details of the RLD block are shown. It comprises:

1- Two memories (RAM 3 and RAM 4).

2- RLD processing circuit.

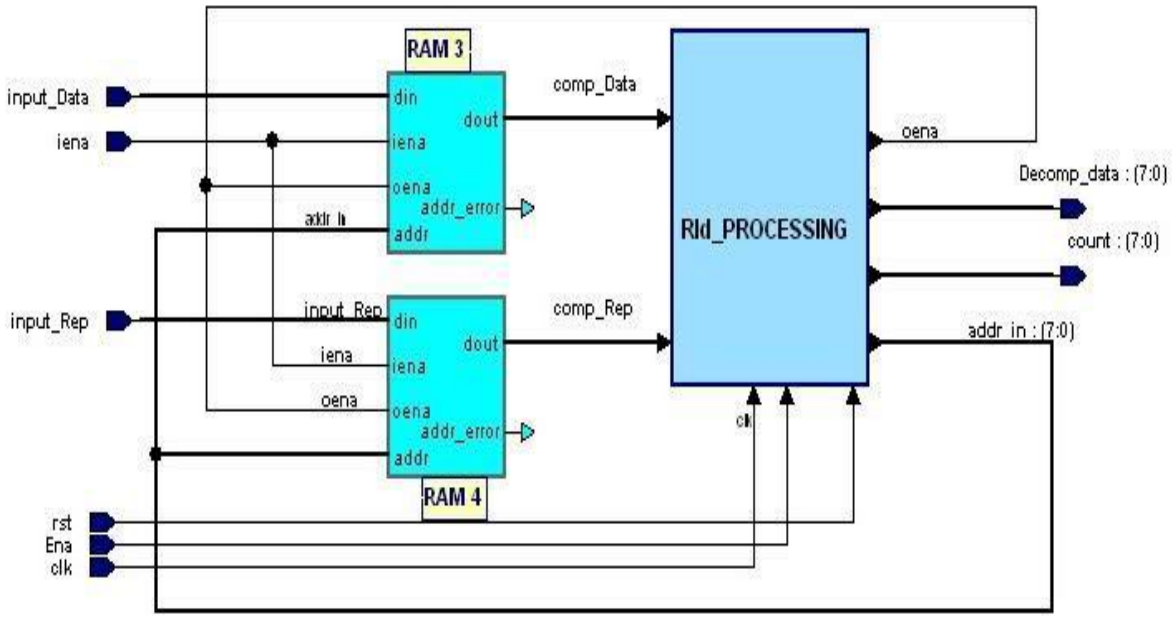

Fig.2. The Block diagram of the RLD design

The compressed data (***input_data*** and ***input_Rep*** are stored in RAM 3 and RAM 4 respectively) and come out when ***oen*** signal becomes '1'. During the storage operation the ***iena*** signal equals '1' and ***oena*** signal becomes '0' i.e., the two memories (RAM 3 and RAM 4) are in writing state after then the two memories becomes in reading state this is when the input data (*input_Data* and *input_Rep*) is written in the last address (i.e., ***addr_in*** =255 or the memories are full of data). At every clock rising edge (***clk***) a new compressed data is entered and stored in RAM 3 and RAM 4. The outputs from RAM 3 and RAM 4 (***comp_data*** and ***comp_Rep,*** respectively) are entered to the RLD processing circuit (*RLD_processing*) in order to perform the decoding process. The ***comp_Rep*** carries the number of data repetition which corresponds the input ***comp_Data***. The RLD processing design procedures are written in VHDL code.

The *RLD_processing* circuit includes a special designed decreasing counter which decrementally counts the input repetition value (***comp_Rep***). At each clock (***clk***) rising edge the counter counts (***count***) are decreased by one and in each decrement count the decompressed data (***Decomp_data***) is coming out from the RLD processing circuit. The ***oena*** signal and ***addr_in*** are the outputs of RLD processing circuit and at the same time the inputs to RAM 3 and RAM 4. The address of RAM 3 is connected to the address of RAM 4 as shown in Fig.2 in order to write ***input_Data*** and ***input_Rep*** with the same clock rising edge at the same address. Once, the counter reaches zero, the ***oena*** signal becomes '1' and the ***addr_in*** increased by one in order to exit a new value form RAM 3 and RAM 4 (***comp_Data*** and ***comp_Rep***) to be entered to the RLD processing circuit (*RLD_processing*) to perform a new decoding process and so on.

3- Simulation results

The functional simulation is done by using **ModelSim 5.5e**. The simulation example is performed on the outputs from RLE design. Assume that the following values are the outputs from the RLE algorithm **(150 8 100 3)**. The input of RAM 3 (***input_Data***) takes the values **150** and **100** (image data or image gray levels values).

The input of RAM 4 (***input_Rep***) takes **8** and **3** (number of repetition of the gray levels values). The output from RLD processing circuit (***Decomp_data***) is **150 150 150 150 150 150 150 150 150 100 100 100 100**. The simulation results are shown in the wave form as illustrated in Fig.3.

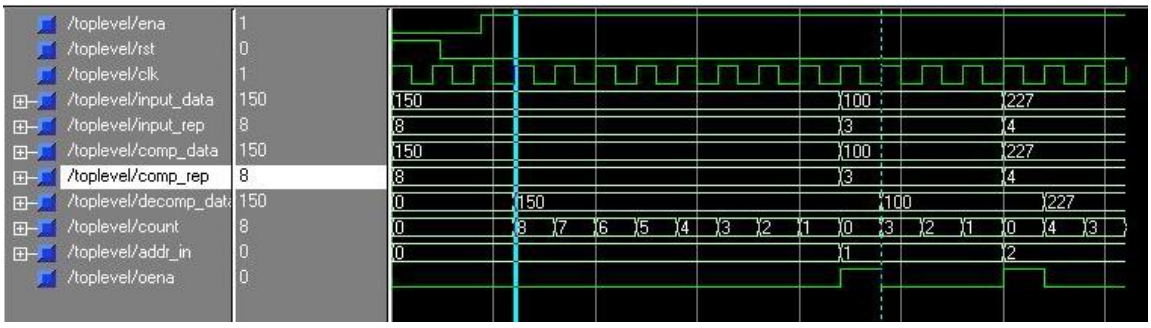

Fig.3. Functional simulation wave forms

The simulation results shows that, the ***oena*** signal equals zero as well as *count* ≠ 0 this means that the RLD processing is still performed on the same input compressed data (***comp_Data*** and ***comp_Rep***). When count reaches zero value the ***oena*** signal becomes '1' and ***addr_in*** increased by one in order to exit a new data value from RAM 3 and RAM 4 to be processed by RLD processing circuit.

The design *synthesis* is done using **Leonardo Spectrum**. *Xilinx-SpartanII-xc2s100pq208* is selected as FPGA chip with clock frequency 20 MHZ [7] in order to download the proposed design. Form the synthesis step, we obtain the area report which shows that **74 CLBs** are used in the design, and the time report which indicates that the output data arrival occurred after **16 n sec** (this means that the time taken to transfer from one input pixel to the next following one is **16 n sec** (ideal simulation). The design is then converted into its gates level to be ready for timing simulation (real simulation).The *place and route* step is done using **Xilinx ISE5.2i** in order to make the design suitable for timing simulation, where time delays between different gates are taken into consideration. After timing simulation of the previous functionally simulated data, it is noticed that, the output decompressed data (***Decomp_data***) is not exiting exactly with the clock (***clk***) rising edge but delayed by **10 n sec** as shown in the wave form which is illustrated in Fig.4. The data arrival time becomes **26 n sec** instead of **16 n sec**, this means that the real output data arrival time occurred after **26 n sec** (not **16 n sec** as in functional simulation).

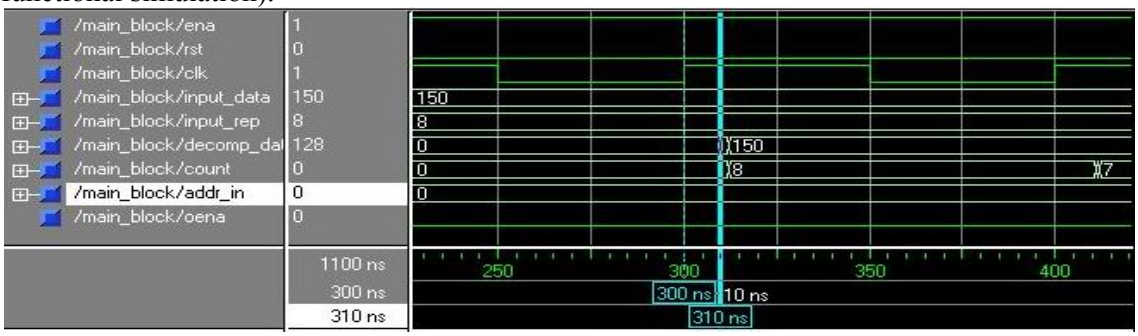

Fig.4. Timing simulation wave forms

After finishing synthesis and timing simulation steps, an **EDIF** (Electronic Digital Interchange Format) file is obtained; this file is used for downloading the design on the selected chip.

4- Testing the RLD design

The FPGA board which include the FPGA chip (*Xilinx-SpartanII-xc2s100pq208*) has a serial port connector (RS232)[7]. Testing the downloaded design is done through the serial port. A design of an UART (Universal Asynchronous Receiver/ Transmitter) should be added to the RLD processing circuit in order to be adapted with the serial port data transfer protocol as illustrated in Fig. 5. The output compressed image data which is obtained from the RLE circuit is transmitted from PC via serial port (bit by bit) to the downloaded UART. The UART converts the received serial data from serial port into parallel data form which is entered to the RLD design where, all inputs and outputs to the RLD design are parallel data (bus of 8-bit). The output parallel data from the downloaded RLD design is converted into serial form by the downloaded UART and then transmitted through the serial port to the PC. The data received by the PC is saved in another file (original image file). It should be noticed that the data included in this file is exactly the same image data (gray levels values) of the original image.

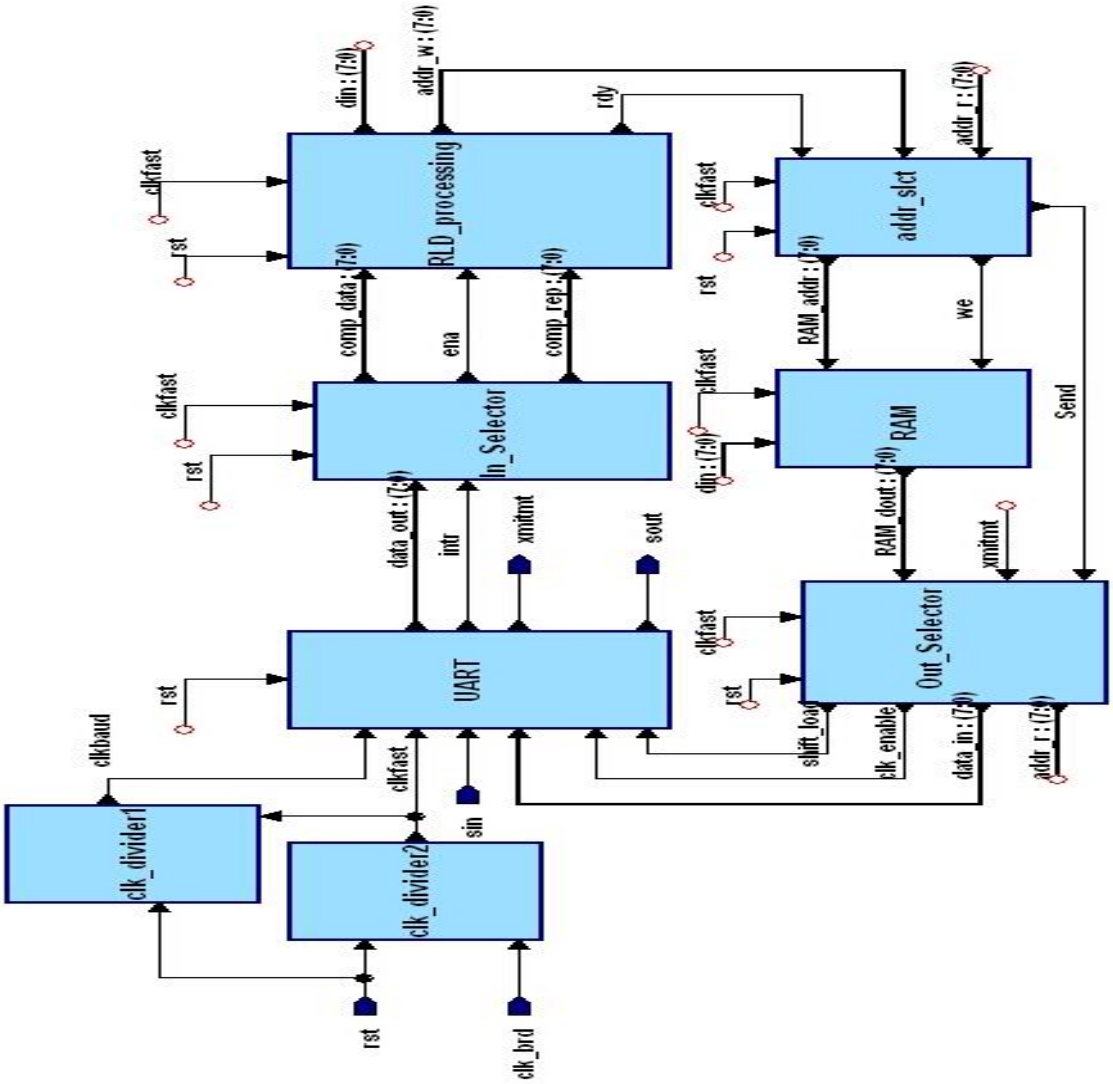

Fig.5. Block diagram of UART design with the RLD processing circuit

The UART consists of receiver (*UART_Rx*) and transmitter (*UART_TX*) as shown in Fig.6. The *UART_Rx* receives the serial data (*sin*) coming from PC via the serial port and converts them into parallel form (***data_out***). The ***data_out*** represents the input data to the input selector (*in_selector*) then to the RLD processing circuit (*RLD_Processing*). A start bit (0) and a stop bit (1) should be added to each eight bit of the input serial data (Pixel value). This means that the frame of input data consists of ten bits (a start bit, a stop bit and the and in between 8-bits which represent the input pixel value). The *intr* signal equals '0' during entering the bits of the pixel value (8-bit) and then converts to '1' at the end of entering the pixel value in order to declare that another pixel value has to be entered. The design procedures of the receiver and transmitter of the UART are written in VHDL code.

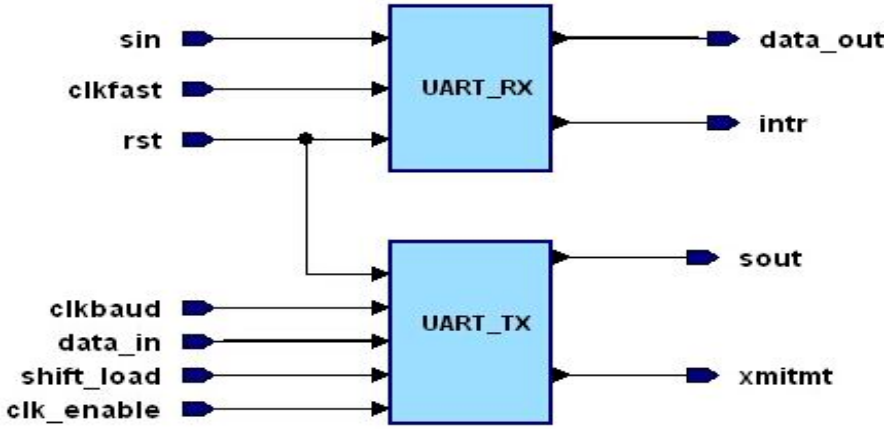

Fig.6. Block diagram of UART Rx / Tx

The rate of data transmission to and from FPGA chip according to the crystal oscillator which is located on the FPGA board is 25 Mbit/sec.[7]. The serial port transmits and receives data to and from FPGA with baud rate of 300 bit/sec., for this reason the clock dividers should be designed to adapt the serial port to be suitable to transmit and receive data. The division of 25 MHZ to be 300 HZ produces fraction number which is so difficult to be simulated and implemented on FPGA so, the crystal oscillator is replaced by another one of 30 MHZ in order to simplify the division process.

The clock divider (*clk_divider2*) receives the clock of FPGA board (***clk_brd***) which is 30 MHZ as illustrated in Fig.5 and converts it to a clock of 4800 HZ (***clkfast***) which is sixteen times faster than the baud rate (300 HZ). The function of ***clkfast*** is to check the correctness of the input received data (*sin*) by the *UART_Rx*. As shown in Fig.7. With each clock rising edge of ***clkfast*** (at least eight rising edge), if the corresponding input data value is '1' then the input data (***sin***) certainly is '1' and vice versa. The ***clkfast*** can be named as data verification clock.

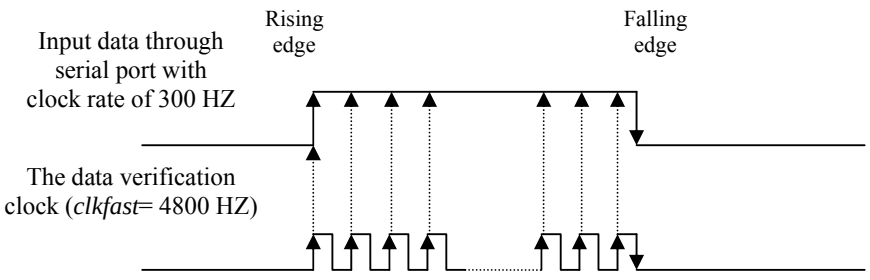

Fig.7. Input data verification through clkfast

The clock divider (*clk_divider1*) divides the ***clkfast*** which is 4800 HZ by 16 to produce the clock Baud (***clkbaud***) which is used for transmitting the output compressed data from the RLE processing circuit through the UART receiver (*UART_Rx*) to the PC via serial port. The design procedures of the clock dividers (*clk_divider1* and *clk_divider2*) are written by VHDL code.

The input selector (*In_Selector*) selects whether the output data (***comp_data***) or their repetitions (***comp_rep***) is first transmitted to the RLD processing circuit (***RLD_pocessing***). It is proposed that the ***comp_data*** is transmitted at first then the ***comp_rep*** is transmitted. The enable signal (***ena***) equals '1' only when ***comp_data*** and ***comp_rep*** are ready for RLD processing circuit i.e., the *RLD_processing* has been received both of ***comp_data*** and ***comp_rep***. The output decompressed image data from the RLD processing circuit is stored in a RAM where, the FPGA chip (*Xilinx-SpartanII-xc2s100pq208*) contains two RAM blocks [7]. The RAM size is proposed to be 256 byte i.e., it can store 256 gray levels values from the decompressed image data.

The address selector circuit (*addr_slct*) selects whether the decompressed image data is written in the RAM or read from the RAM. The RAM address (*RAM addr*) takes the value of *addr_w* when the RAM is in writing state and in this case the write enable signal (*we*) becomes '1'. The RAM address takes the value of *addr_r* when the RAM is in reading state and in this case the *we* signal becomes '0'. The address selector works only when ready signal (*rdy*) equals '1' which means that the output decompressed data (*din*) from the RLD processing circuit is ready to be written in the RAM. The signal *Send* equals '0' in case of writing the decompressed data in the RAM (*we*='1') and equals '1' in case of reading the decompressed data from the RAM (we='0').

The output selector (*Out_Selector*) works only when *send* signal equals '1', the *send* signal equals '1' to read the decompressed data (***RAM_dout***) from the RAM. The UART transmitter serially transmits the output decompressed data which stored in the RAM addresses (256 addresses) via serial port to the PC, i.e., **sout** represents the output decompressed data in a serial form. The *data_in* represents the output data from the output selector which entered to the *UART_Tx* and loaded in a register when ***shift_load*** signal equals '0' and ***clk_enable*** equals '1' (***clk_enable*** is considered the enable signal of the UART transmitter). The ***shift_load*** signal converts to '1' during data transmission process via a serial port. The ***clk_enable*** and ***shift_load*** signals become '0' at the end of each data transmission. The ***xmitmt*** signal works as a flag where, it equals '0' through data transmission from *UART_Tx* to PC and equals '1' at the end of each data transmission, in this case the ***addr_r*** is increased by one in order to read the next data stored in the RAM.

5- Conclusion

From the area report and time report obtained from the synthesis step, the hardware utilization of the selected FPGA chip is given in Table 2 and the real data arrival time is given in Table 3.

Table 2. FPGA area report for RLD design

	Used	Available	Utilization
CLBs	74	1200	6.17%
IOs	44	140	31.43
Function Generators	133	2400	5.45%
Dffs or Latches	32	3120	1.02%

Table 3. FPGA time report after timing simulation for RLD design

Data required Time (n sec.)	50
Data arrival time (n sec.)	26
Slack (n sec.)	24

From are report, it is clear that the RLD design fits the downloaded target device (*Xilinx-SpartanII-xc2s100pq208*) since, the design doesn't exceed the available hardware resources (CLB_s, IO_s, Dff_s, and function generators).

From the time report, it is clear that the RLD design has a positive slack time equals to **24 n sec**. This means that the downloaded design on the selected FPGA chip (*Xilinx-SpartanII-xc2s100pq208*) can work properly until **41.7** Mhz without errors in output values.

5- References:

[1] Rafel C.Gonzalez , Richard E.Woods, "*Digital image processing*", Second edition, university of Tennesse, MedData Interactive, (2002).

[2] Guy E.Blelloch ,"*Introduction to Data compression* ", computer science department, Carnegie Mellon University, October 16,(2001).

[3] Gouda Ismail Salama, Fawzy Eltohamy Hassan, Mohammed Sharawy Ibrahim, Ramy Mohammed Bahy, "An Implementation of the Run-Length Encode Algorithm using FPGA ", 11TH international conference on Aerospace Science & Aviation Technology, Military Technical College, Kobry El-Koba, Cairo, Egypt, 17-19 May 2005.

[4] Tom Kaminski, "A Hardware Implementation of Arithmetic Compression", Final Report for 24.433 Thesis Project for the faculty of Computer Engineering at the University of Manitoba",March, (2001).

[5] R M Susilo, T R Bretschneider, "On Real time Satellite Image Compression of X-sat", School of computer engineering, Nanyang Technological University, 2003.

[6] Desigining with FPGA Advantage, Mentor Grahic, student workbook, software V5.2, January (2002).

[7] Memec Spartan II "LC Users Guide V1.0", July 21, (2003).