

Hardware Implementation of Densely Packed Decimal Encoding

An optimized approach supporting run-time user input

By

M.S.S.A.Raghuveer

Roll Number : 108CS077

B.Tech VIII Semester 2012

May 14, 2012



**National Institute of Technology
Rourkela, Odisha - 769008**



National Institute of Technology Rourkela, Odisha

Certificate

This is to certify that the work in the thesis entitled Hardware Implementation of Densely Packed Decimal Encoding - An optimized approach supporting run-time user input submitted by M.S.S.A.Raghuveer is a record of an authentic work carried out by him under my supervision and guidance in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering at National Institute of Technology, Rourkela.

Place : NIT Rourkela

Date :

Dr. Ashok Kumar Turuk
Associate Professor
Head of the Department
Computer Science and Engineering
NIT Rourkela

Acknowledgements

I would like to express my sincere gratitude to my supervisor Dr.Ashok Kumar Turuk,Department of Computer Science and Engineering for his able guidance,wise suggestions and the never ending support and faith,without which my work would not have been possible.

I would also like to extend my thanks to Prof.A.K.Swain,Department of Electronics and Communication Engineering and other senior members the VLSI lab for their cooperation to my project work.

M.S.S.A.Raghuveer

(108CS077)

Abstract

BCD Encoding scheme represents each Decimal digit(Base 10) by its own binary sequence of 4 bits.Though this scheme remains highly useful for storage and simple operations on decimal data,compact representations hold more significance in some applications.An encoding scheme was proposed by Chen and Ho named "Chen-Ho Encoding".This encoding represents a three digit decimal in 10 bits unlike BCD which requires 12 bits,thus giving more efficiency and less wastage.This uses an algorithm which uses simple boolean operations to compress the 12 BCD bits into 10 and also reverse the process[1].

DPD encoding is an improvisation of Chen-Ho encoding scheme.This overcomes the limitation of Chen-Ho encoding which requires the decimal number to be a multiple of 3 digits[2].This codes arbitrary length decimal numbers as 10 bits.This enables the best use of available resources like storage space and hardware registers.BCD encoding results in high wastage of bit-pattern space.The objective of DPD compression is to use this space for a long string of digits.

This thesis embodies the work done to implement an optimized Densely Packed Decimal (DPD) encoding on hardware using VHDL and Xilinx Spartan 3E FPGA.

Contents

1	Introduction	5
2	Literature Survey	6
2.1	Binary Coded Decimal(BCD) Encoding	6
2.2	Chen-Ho Encoding	7
2.3	DPD Encoding	8
2.3.1	Details of encoding	8
2.3.2	Proposed Hardware Logic	9
2.3.3	Optimized Compressor Module	11
3	Implementation	14
3.1	Simulation of the Compressor Module with BCD inputs	14
3.2	Simulation of Compressor Module with Binary inputs	14
3.2.1	Binary to BCD Conversion	14
3.3	Implementation on FPGA with fixed inputs	19
3.4	Implementation on FPGA with run-time user inputs	20
3.4.1	UART	20
3.4.2	UART Receiver Subsystem	21
3.4.3	Input Appending Module	23
3.4.4	Output Appending Module	25
3.4.5	UART Transceiver System	25
3.5	HyperTerminal	26

4 Conclusion	27
---------------------	-----------

List of Figures

1	BCD Encoding	6
2	Encoding/Compression	10
3	Decoding/Expansion	10
4	Simulation of Compressor Module	15
5	Block Diagram of Binary to BCD Converter	17
6	Simulation of Binary to BCD converter	18
7	Block Diagram of UART Receiver	22
8	Flow Chart of UART Receiver	24
9	Flow Chart of UART Receiver(contd)	24

List of Tables

1	Comparision of Encoding Schemes	9
---	---	---

1 Introduction

The increasing usage of decimal data in various real time applications has led the researchers to pursue a thorough work on various techniques for the implementation of Decimal Floating-Point Arithmetic. Decimal Floating-Point Arithmetic holds high importance for its ability to represent decimal fractions exactly and to mimic manual calculations that perform decimal rounding. Binary Floating-Point Arithmetic can neither provide exact decimal rounding nor can represent many decimal fractions exactly. The software implementation of Decimal Floating-Point Arithmetic eliminates these errors. However it is highly slower than the Binary Floating-Point Arithmetic operations. Hence hardware implementation of DFP Arithmetic is expected to keep up the ever growing demands of the industry. The inclusion of specifications for DFP Arithmetic in IEEE Draft Standard for Floating-point Arithmetic (IEEE P754) and in IEEE-754-2008 standard depicts the prominence of this area.

The IEEE 754-2008 specifies two encodings of decimal numbers corresponding to the implementations of decimal arithmetic. Binary encoding allows for efficient software operations, using native binary operations of a processor. Decimal encoding (or DPD) was designed to make a hardware implementation of decimal floating-point arithmetic reach the desired goals. In this format, significand is encoded in a group of 3 digits, each group in 10 bits.

2 Literature Survey

2.1 Binary Coded Decimal(BCD) Encoding

BCD notation is one of the highly used notations of a decimal number because of its simplicity. It requires a 4-bit binary pattern to represent a digit from 0 to 9 (Figure 1). The 4-bit unit which stores the BCD digit is named a 'nibble'.

For example, the BCD notation for the number 192 is 0001 1001 0010 whereas the pure binary notation is 11000000.

Decimal Digit	BCD 8 4 2 1
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1

Figure 1: BCD Encoding

However much of the bit-pattern space is wasted by this notation. For example consider a 3 digit decimal. It requires 12 bits to be represented in this format. But 12 bits can represent 4096 patterns in binary whereas in decimal 3 digits can represent only 1000 numbers. Thus the rest of 3096 patterns are wasted (76

Hence a compression to squeeze out the unused space for efficient usage is recommended. If 10 bits are used for representation (since $1024 > 1000$), the wastage can be reduced from 76% to approximately 2.5%. Two such encoding schemes that do this are Chen-Ho Encoding and Densely Packed Decimal (DPD) Encoding. The latter has the advantage that it represents a 2 digit number in an optimal 7 bits and a single digit number in 4 bits as in BCD.

2.2 Chen-Ho Encoding

The Chen-Ho Encoding represents 3 decimal digits in 10 bits that can depict upto 1024 patterns and therefore can encode the 1000 possibilities for 3 digits in decimal system ensuring little waste of bit-pattern space.

Chen-Ho is better than the straight forward 10 bit binary representation because it involves only simple boolean operations to convert from/to BCD format. This has an advantage over the variable-length schemes, because its fixed length mapping allows simpler encoding and decoding logics.

This scheme works well when the decimal number has number of digits as multiples of 3. It is less desired for other lengths because either digits must be wasted or more than one encoding schemes have to be adopted to represent the other digits of the number.

2.3 DPD Encoding

This scheme was first proposed by M.F.Cowlshaw and is an improvement over the Chen-Ho encoding. It uses the coding scheme equivalent to the Chen-ho but instead of using the Huffman code it uses a fresh arrangement of bits thus obtaining some prime advantages over the Chen-Ho scheme as follows:

1. The limitation that the number of decimal digits must be a multiple of three is overcome. This can encode any arbitrary number of decimal digits. This represents a 2 digit number in an optimal 7 bits and a single digit number in 4 bits.
2. The encoding decimal numbers can be expanded into a longer unit just by padding with zero bits and no re-encoding is necessary, unlike the Chen-Ho encoding which requires a complete re-encoding.
3. The decimal numbers in range 0 through 79 have the same right-aligned notations as in BCD. In Chen-Ho encoding only numbers in the range 0 through 7 have the same notations as in BCD.

The above advantages clearly depict the fact that DPD Encoding is a better choice than Chen-Ho Encoding for both hardware and software implementations. Here are some comparative examples of various encoding schemes discussed above:

2.3.1 Details of encoding

This encoding scheme categorizes each of the three digits as follows:

Decimal	BCD	Chen-Ho	DPD
005	0000 0000 0101	000 000 0101	000 000 0101
009	0000 0000 1001	110 000 0001	000 000 1001
055	0000 0101 0101	000 010 1101	000 101 0101
099	0000 1001 1001	111 000 1001	000 101 1111
555	0101 0101 0101	010 110 1101	101 101 0101
999	1001 1001 1001	111 111 1001	001 111 1111

Table 1: Comparison of Encoding Schemes

- Small(0-7,requires 3 bits)
- Large(8 Or 9,requies 1 bit)

The most significant bit of each BCD digit is 0 for small values and 1 for large values. In the following explanations,

- BCD representation of 3 digits are abcd,efgh,ijkm
- DPD representation of 3 digits are pqr,stu,v,w

The Encoding and Decoding particluars are shown in the following figures:

2.3.2 Proposed Hardware Logic

M.F.Cowlshaw[1] proposed the following logics for the hardware design of the DPD encoder:

- $p = (b \text{ I\& } a) \text{ — } (j \text{ I\& } a \text{ I\& } i) \text{ — } (f \text{ I\& } a \text{ I\& } e \text{ I\& } i)$

aei	pqr stu v wxy
000	bcd fgh 0 jkm
001	bcd fgh 1 00m
010	bcd jkh 1 01m
100	jkd fgh 1 10m
110	jkd 00h 1 11m
101	fgd 01h 1 11m
011	bcd 10h 1 11m
111	00d 11h 1 11m

Figure 2: Encoding/Compression

vwkst	abcd efgh i jkm
0....	Opqr Ostu Owxy
100..	Opqr Ostu 100y
101..	Opqr 100u Osty
110..	100r Ostu Opqy
11100	100r 100u Opqy
11101	100r Opqu 100y
11110	Opqr 100u 100y
11111	100r 100u 100y
('.' means don't-care.)	

Figure 3: Decoding/Expansion

- $q = (c \text{ I\& } a) \text{ --- } (k \text{ I\& } a \text{ I\& } i) \text{ --- } (g \text{ I\& } a \text{ I\& } e \text{ I\& } i)$
- $r = d$
- $s = (f \text{ I\& } e \text{ I\& } (a \text{ I\& } i)) \text{ --- } (j \text{ I\& } (a \text{ I\& } e \text{ I\& } i)) \text{ --- } (e \text{ I\& } i)$
- $t = (g \text{ I\& } e \text{ I\& } (a \text{ I\& } i)) \text{ --- } (k \text{ I\& } (a \text{ I\& } e \text{ I\& } i)) \text{ --- } (a \text{ I\& } i)$
- $u = h$

- $v = a \text{ --- } e \text{ --- } I$
- $w = a \text{ --- } (e \text{ I\& } i) \text{ --- } (j \text{ I\& } e \text{ I\& } i)$
- $x = e \text{ --- } (a \text{ I\& } i) \text{ --- } (k \text{ I\& } a \text{ I\& } i)$
- $y = m$

2.3.3 Optimized Compressor Module

The compressor module proposed by M.F.Cowlshaw takes 4 gate delays for completion. However by analyzing each signal at a time separately from Figure.2 3 logic gate delay can be achieved using pipelining technique[3][4].

The details of each of the output signals analyzed separately are as follows:

Logic Module for signal p

$$s1 \text{ j= } b \text{ and not } a;$$

$$s2 \text{ j= } j \text{ and } a \text{ and not } i;$$

$$s3 \text{ j= } f \text{ and } a \text{ and not } e \text{ and } i;$$

$$p \text{ j= } s1 \text{ or } s2 \text{ or } s3 ;$$

Logic Module for signal q

$$s1 \text{ j= } b \text{ and not } a;$$

$$s2 \text{ j= } k \text{ and } a \text{ and not } i;$$

$$s3 \text{ j= } g \text{ and } a \text{ and not } e \text{ and } i;$$

$$q \text{ j= } s1 \text{ or } s2 \text{ or } s3 ;$$

Logic Module for signal s

$s1 \text{ } j = f \text{ and not } e \text{ and not } i;$

$s2 \text{ } j = f \text{ and not } a \text{ and not } e;$

$s3 \text{ } j = j \text{ and not } a \text{ and } e \text{ and not } i;$

$s4 \text{ } j = e \text{ and } i;$

$s \text{ } j = s1 \text{ or } s2 \text{ or } s3 \text{ or } s4 ;$

Logic Module for signal t

$s1 \text{ } j = g \text{ and not } a \text{ and not } e;$

$s2 \text{ } j = g \text{ and not } e \text{ and not } i;$

$s3 \text{ } j = k \text{ and not } a \text{ and } e \text{ and not } i;$

$s4 \text{ } j = a \text{ and } i;$

$t \text{ } j = s1 \text{ or } s2 \text{ or } s3 \text{ or } s4;$

Logic Module for signal w

$s1 \text{ } j = j \text{ and not } a \text{ and not } e \text{ and not } i;$

$s2 \text{ } j = e \text{ and } i;$

$w \text{ } j = s1 \text{ or } s2 \text{ or } a;$

Logic Module for signal x

$s1 \text{ } j = k \text{ and not } a \text{ and not } e \text{ and not } i;$

$s2 \text{ } j = a \text{ and } i;$

$x \text{ } j = s1 \text{ or } s2 \text{ or } e;$

Logic Module for signal r,u,y

$r_i = d;$

$u_i = h;$

$y_i = m;$

3 Implementation

The implementation of the optimized DPD system with run-time user input is done in an incremental manner as described below.

3.1 Simulation of the Compressor Module with BCD inputs

In the first step, testbenches were developed for the optimized compressor module[2]. Different inputs as BCD sequences each of 12 bits were taken and fed sequentially to the module and the outputs were recorded. Simulations were done using Xilinx ISE 10.1. Figure 4 shows the result of the simulation:

3.2 Simulation of Compressor Module with Binary inputs

In the second step, a more closer real life situation is considered. A decimal number (in Binary form) is fed as input to the compressor module and simulations are done.

3.2.1 Binary to BCD Conversion

The decimal number input fed to the compressor module needs to be converted to BCD format and then the output of the conversion module (BCD sequence) is fed as input to the compressor.

The conversion process is implemented by simple algorithm implemented in VHDL[3]. This takes the 8 bit binary number as input and gives 12 bit BCD

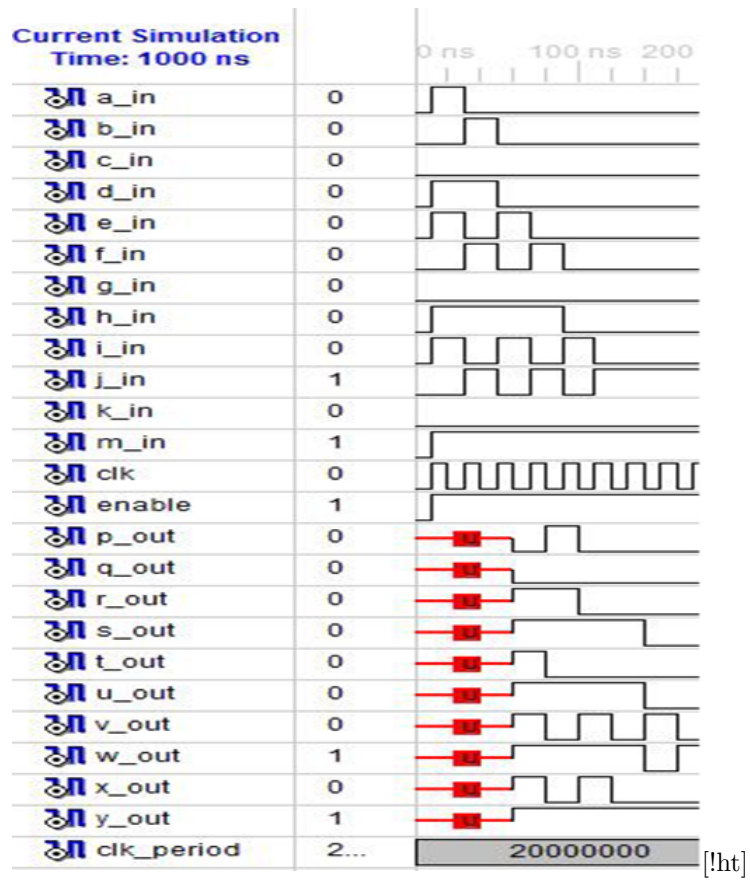


Figure 4: Simulation of Compressor Module

sequence as output. The algorithm follows the following steps:

1. The binary number is shifted to left by 1 bit
2. Three columns Units, Tens and Hundreds are defined each of 4 bit width from left to right.
3. If value of a number in any of the BCD columns is 5 or greater, 3 is added to it.
4. The first step is repeated till the entire binary number is shifted.

Figure 5 shows the block diagram of the Binary to BCD converter. The following is an example of the conversion process with 11110011 as binary input explaining

the algorithm.	0000 0000 0000 11110011	Initialization
0000 0000 0001 11100110	S1	
0000 0000 0011 11001100	S2	
0000 0000 0111 10011000	S3	
0000 0000 1010 10011000	Added 3 to 7	
0000 0001 0101 00110000	S4	
0000 0001 1000 00110000	Added 3 to 5	
0000 0011 0000 01100000	S5	
0000 0110 0000 11000000	S6	
0000 1001 0000 11000000	Added 3 to 6	
0001 0010 0001 10000000	S7	
0010 0100 0011 00000000	S8	

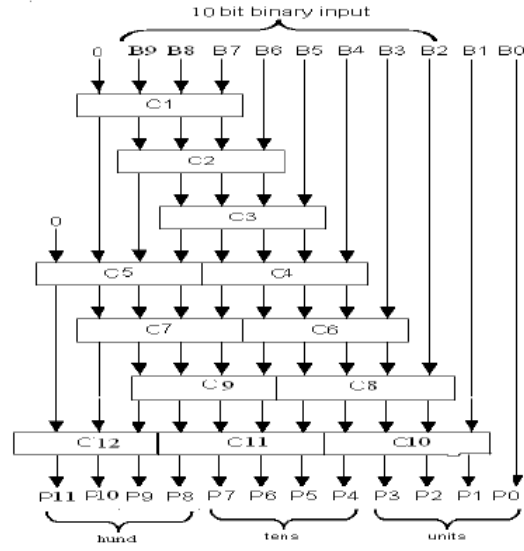


Figure 5: Block Diagram of Binary to BCD Converter

This converter is implemented before the compression process is taken place. The simulation of Binary to BCD conversion is shown in Figure 6.

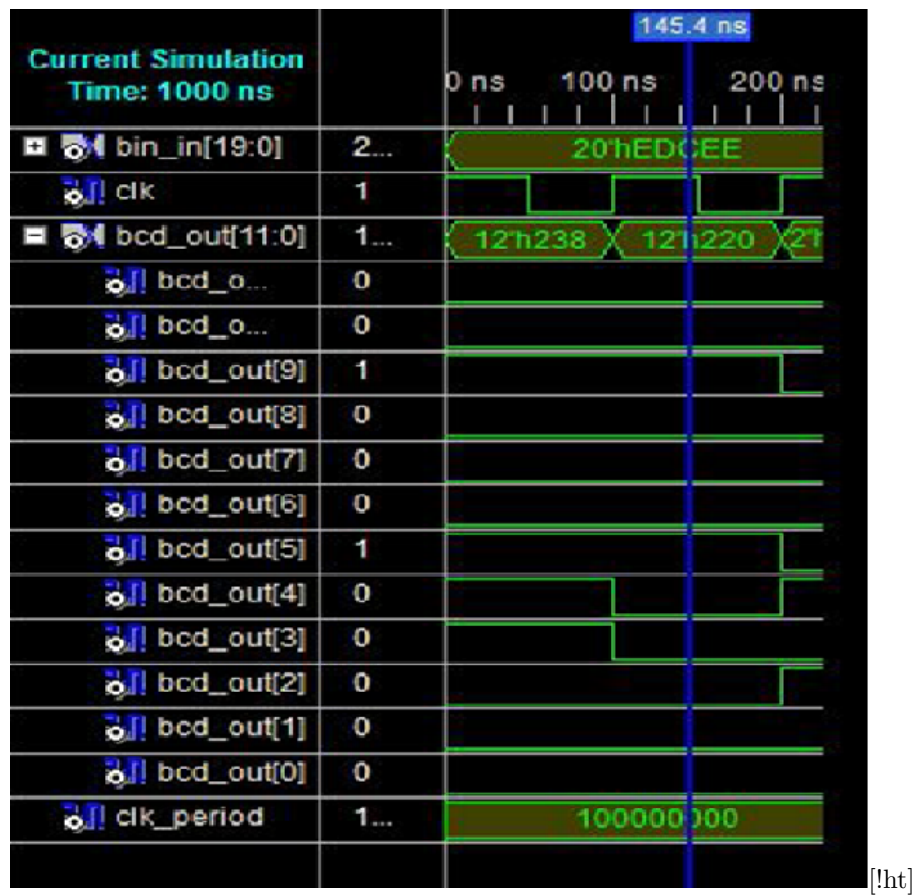


Figure 6: Simulation of Binary to BCD converter

3.3 Implementation on FPGA with fixed inputs

AN FPGA is an aronym for Field Programmable Gate Array.As the name suggests,this is an integrated circuit that can be programmed/configured by the customer or the designer as per the requirements of the logic design that is to be implemented.Implementing a logic design on a FPGA usually includes the following steps:

1. The description of the desired logic circuit is entered and the FPGA is configured in some hardware description language like VHDL or Verilog.In this project,VHDL is used to code the logic design proposed to be implemented for the compressor module and other functions.
2. A logic Synthesizer program is used to transform the HDL into a 'Netlist'.The netlist is just a description of the various logic gates in your design and how they are interconnected.
3. Implementation tools are used to map the logic gates and inter-connections into the FPGA.The FPGA consists of many Configurable Logic Blocks,that can be decomposed further into Look-Up Tables.The CLBs and LUTs are interwoven with various Routing Resources.The 'Mapping' tool collects the netlist gates into groups that fit into the LUTs and then the 'Place and Route' tool assigns the groups to specific CLBs.
4. After the implementation step is finished,a program extracts the state of the switches in the routing matrices and generates a bitstream where the ones and zeroes correspond to open or closed switches

5. The bitstream is downloaded into a physical FPGA chip).The electronic switches in the FPGA open or close in response to the binary bits in the bitstream. Upon completion of the downloading, the FPGA will perform the operations specified by your HDL code or schematic

Here we used the Xilinx Spartan 3E FPGA(Figure 7) for the implementation of the modules described before.The VHDL code for the Binary to BCD conversion and BCD to DPD compression were implemented on this FPGA after proper synthesis,design implementation and constraint definitions.Fixed test inputs were given to the synthesized code and the outputs were verified.

3.4 Implementation on FPGA with run-time user inputs

3.4.1 UART

UART is an acronym for Universal Asynchronous Receiver-Transmitter.It is a circuit that sends parallel data through serial line.UARTs are frequently used in conjunction with the EIA (Electronic Industries Alliance) RS-232 standard, which specifies the electrical, mechanical, functional, and procedural characteristics of two data communication equipment.[6] The Spartan 3E board has a RS232 port with a standard 9-pin connector.The board contains The necessary voltage converter chip is present on the board which configures the various RS-232s control signals and automatically generates an acknowledgment for the PCs serial port.

The UART contains a transmitter and a receiver. The transmitter is a shift register that loads parallel data and shifts it out bit by bit at a predefined rate. The job of the receiver is just the opposite. When the serial is idle, it maintains binary '1'. The transmission starts with a start bit which is generally '0' followed by the data bits and then ends with stop bits at '1'. To check errors parity bits may also be included in the transmission sequence.

No clock signals are transmitted through the serial line. Hence it is named 'asynchronous'. Before the transmission starts, both the transmitter and receiver agree upon a set of parameters like Baud Rate (Number of bits transmitted per second), number of data bits (can be 6, 7, 8), stop bits and parity bits.

Here we use a Baud Rate of 19200 for a clock of 50MHz frequency. The serial line parameters include one start bit(0), 8 data bits and one stop bit(1). No techniques for flow control or error control are included. The implementation of UART integrated compressor module is performed in five phases

3.4.2 UART Receiver Subsystem

As no clock signals are transmitted through the serial line, the receiver takes in the data bits according to the predefined parameters described above. Oversampling scheme is used to estimate the middle points of transmitted bits and they are retrieved at those points correspondingly.

The sampling rate used here is 16 times the Baud Rate(19200). This means that each serial bit is sampled 16 times. This scheme basically performs the

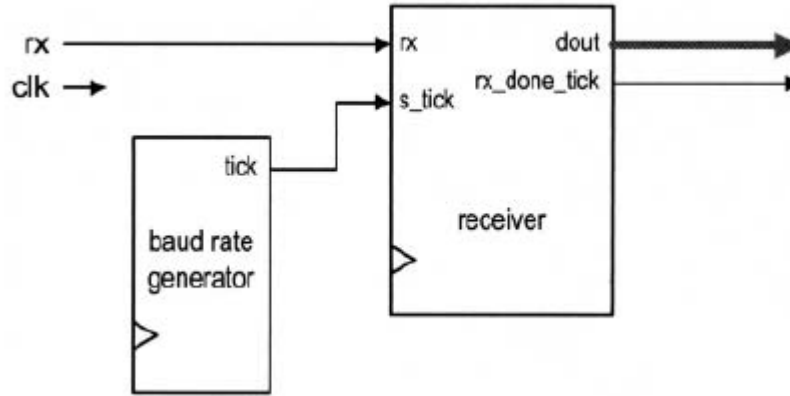


Figure 7: Block Diagram of UART Receiver

function of a clock. Instead of considering the rising edge of the clock to accept an input bit, the number of ticks are monitored. However, this scheme is not suitable for high data rates [5]. The block diagram of the UART receiver subsystem is shown in Figure 7.

In the above figure, Receiver is the circuit that accepts the data bits through the oversampling procedure stated above. Baud Rate Generator is the circuit that produces the sampling ticks.

mod-M Counter Here we use the mod-M counter as the Baud Rate Generator. Mod-M Counter counts from 0 to (M-1) and then wraps around. This requires two values as parameters

- M is the value to which it should count upto
- N is the number of bits required to count to the limit

The working of the receiver subsystem is shown in Figure 8 and Figure 9.

- Four states of working are defined Idle,Start,Data and Stop.When the serial line is high at '1' it is in the idle state.When the 'rx' bit becomes '0' the system enters the Start state.Then depending upon the value of 's' register the data is transmitted and the system enters the Stop state when 'n' register reaches the limit of the number of data bits to be transmitted.
- Three sets of registers s,n,b are considered for three functions storing the value of tick,number of bits transmitted and transmitted bit sequence respectively.

3.4.3 Input Appending Module

The UART Receiver subsystem receives the data bits following the process explained above.RS232 communication occurs in ASCII format(using HyperTerminal discussed later).A case map is adopted to obtain the corresponding BCD value for every ASCII value that is received.Thus for every 8 data bits received as ASCII the case map gives out 4 bits that are in BCD.

The compressor module accepts the bit sequence obtained from the Receiver subsystem only when the sequence has 12 bits.It then compresses these 12 bits to 10 DPD bits.Hence the Input Appending Module acts as a buffer between the Receiver system and the Compressor module.It takes in the 4-bit BCD sequences from Receiver and append them with each other sequentially and gives out the 12-bit sequence to Compressor Module.This module also takes

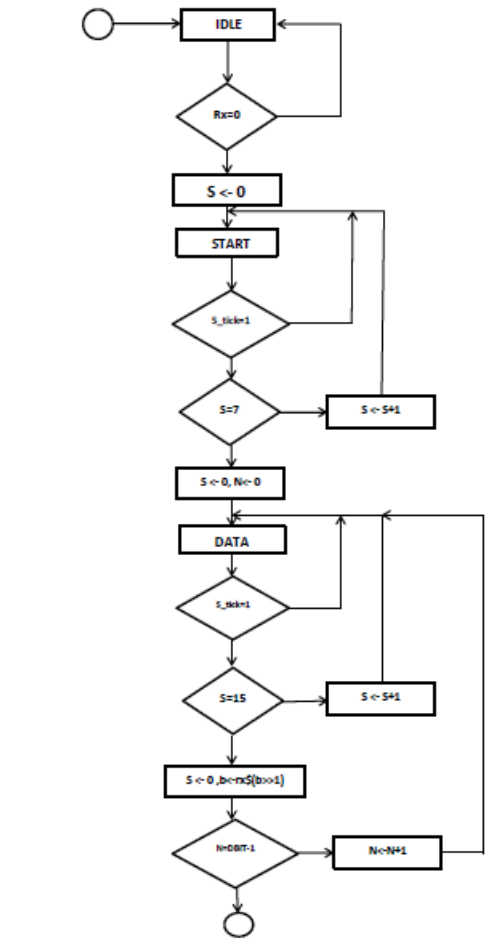


Figure 8: Flow Chart of UART Receiver

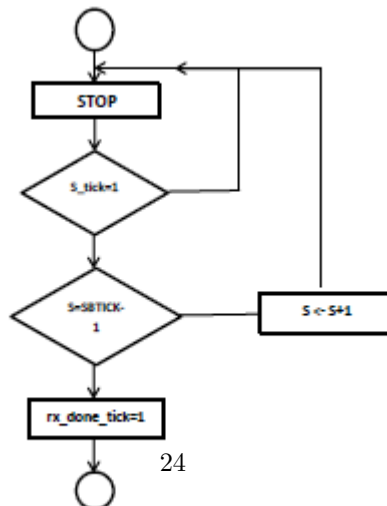


Figure 9: Flow Chart of UART Receiver(contd)

care if the data transmission ends before a 12-bit sequence is made. It appends the received sequence with necessary zeros and gives it to the compressor.

3.4.4 Output Appending Module

The core function of the Output appending module is same as the input appending module. The compressor module receives 12-bit BCD sequences from the Input Appender Module and compresses it to 10-bit DPD sequence. But the data transmission is carried out according to some predetermined parameters as explained before. One of them is the number of data bits transferred (8 bits here).

This module takes in 10-bit DPD sequences from the compressor module and makes the necessary adaptations to make it satisfy the parameters defined.

3.4.5 UART Transceiver System

The organization of the transmitter system is same as the receiver system. The UART transmitter is essentially a shift register that shifts out data bits at a specific rate. The shift rate is controlled by one-clock-cycle Enable ticks generated by the Baud Rate Generator. Generally Transmitter system shares the same Baud Rate Generator as the Receiver system. A data bit is shifted out every 16 enable ticks. The working chart of the Transmitter system is similar to the Receiver system (Figure 8).

3.5 HyperTerminal

On the user side,HyperTerminal of Windows was used.Windows' HyperTerminal is an application that can be used as a virtual terminal to interact with the Spartan 3E board.The HyperTerminal connection is configured according to the predetermined parameters.It uses the COM port of the PC to make a connection with the FPGA.

Dynamic input from the user can be taken using this virtual terminal and the processed data(After Compression) can be loop back to be shown on the same window.

4 Conclusion

The optimized design of the compressor module is simulated with fixed inputs, both BCD and Binary. The VHDL code is also implemented on FPGA after proper synthesis, constraint definitions and design implementation. Implementation with Run-Time user input is also performed and verified.

References

- [1] Tien Chi Chen and Irving T. Ho. Storage efficient representation of decimal data. CACM, 18(1):49-52, January 1975.
- [2] M. F. Cowlishaw. Densely packed decimal encoding. IEEE Proceedings Computers and Digital Techniques, 149(3):102-104, May 2002.
- [3] L. Eisen, J. W. Ward, H.W. Tast, N. Mading, J. Leenstra, S. M. Mueller, C. Jacobi, J. Preiss, E. M. Schwarz, and S. R. Carlough. Ibm power6 accelerators: Vmx and dfu. IBM Journal of Research and Development, 51(6):1-21, November 2007.
- [4] Kamlesh Sulanki. Pipelined Implementation of Densely Packed Decimal Encoding. 18 May 2011. <http://ethesis.nitrkl.ac.in/2555/>
- [5] Cowlishaw, M. F., "Summary of Densely Packed Decimal encoding". <http://speleotrove.com/decimal/DPDecimal.html>.
- [6] Pong P. Chu. *FPGA prototyping by VHDL Examples-Xilinx Spartan 3 Version*. Wiley Publications 2008
- [7] Douglas L. Perry. VHDL Programming : By Example McGraw-Hill Publications 4th Edition
- [8] ISE Simulator In-depth Tutorial. UG682(v1.0) April 27, 2009
- [9] Spartan-3E Starter Kit Board User Guide. UG230 (v1.0) March 9, 2006
- [10] <http://www.xilinx.com/forums>