A
Dissertation
On

REAL TIME TRANSFER OF DATA FROM RING LASER GYRO USING BLUETOOTH

Submitted in partial fulfillment of the requirements for the award of
Degree of

BACHELOR OF TECHNOLOGY

In

ELECTRONICS AND COMMUNICATION ENGINNEERING

By

MULA DIWAKAR BABU	190040346
MANNE PRANETHA	190040311
SAHITHA MEDASANI	190040326
SAI AMARESH DOGIPARTHI	190040121

Under the guidance of

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
KL UNIVERSITY
Green Field, Vaddeswaram, Vijayawada, A.P– 522 502

2021

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
KL UNIVERSITY
Green Field, Vaddeswaram, Vijayawada, A.P– 522 502

CERTIFICATE

This is to certify that the project entitled

REAL TIME TRANSFER OF DATA FROM RLG USING BLUETOOTH

Is the bonafide work of

MULA DIWAKAR BABU	190040346
MANNE PRANETHA	190040311
SAHITHA MEDASANI	190040326
SAI AMARESH DOGIPARTHI	190040121

Submitted in partial fulfillment of the requirement for the award of the degree of Bachelor of Technology in Electronics and Communication Engineering at Bandari Srinivas Institute of Technology is a record of bonafide work carried by them under out guidance. The matter contained in this dissertation has not been submitted in any other university for the award of any degree or diploma

ABSTRACT

Ring laser gyro is a device which is used to measure angles very accurately and hence is extensively used in air crafts, naval systems and strategic missiles systems for navigational purposes. A set of three gyros and a set of three accelerometers are used to make one inertial navigational system which can locate any change of the vehicle from place to place. When these gyros are integrated in the inertial navigation system (INS) it is not possible to open the gyros and measure their status of working. Only through some wire link minimum data can be sent till the launch of the misile system.

It is going to help if there is a wireless communication between the gyro and the external world (computer) so that entire data of the gyro can be monitored till the last moment with out effecting the performance of the gyro and communication can also be established between the gyros effectively and the performance of the INS can be enhanced with better information flow. At the same time it is important to secure vital data of the gyro. Hence wireless communication with long range is not recommended.

It is planned to utilize the relatively new technology of BLUE T0OTH protocol for communicating the gyro output to the external world replacing the existing parallel port output RS 422. This involves changing the configuration of FPGA where the communication protocol is presently residing and also using a blue tooth device to transmit to ranges of the order of 10mts non stop

In this project it is proposed to study about ring laser gyro device and also the way the gyro output is configured in the FPGA. It is proposed to understand the relatively new Lechnology of blue tooth and the devices available to configure the gyro hardware without affecting the present setup Hardware and software will be designed and realized to establish the blue tooth communication between gyro and the notebook pe where built in Bluetooth transceiver is present. Existing display software will be utilized to display the data acquired through blue tooth device.

CONTENTS

List of Figures

CHAPTER I

INTRODUCTION

Aim of the project:

Configuring an FPGA for transferring live data from existing Ring Laser Gyro to PC using Bluetooth.

Introduction To The Project

Systems using wireless technology are more accessible than that of wired.

So, most of the systems are shifting from wired to wireless. There are many techniques of transferring data wirelessly one of that is BLUETOOTH protocol.

The communication between gyro and PC is through wires, for the first time in this project we are using a wireless technology.

The purpose of this project is used to transfer a data using wireless communication technology between a Ring Laser Gyro and a personal computer. Here BLUETOOTH technology is introduced for both the devices to connect communicate wirelessly. The technology of BLUETOOTH is required to transmit the data in wireless mode for shorter ranges with security. Presently only wired communication exists. So, for the first time Wireless communication is used to enable the gyro.

In the project FPGA has been configured using VHDL to design a receiver transmitter for a serial data port and enabled a COM port for communication with 'C' programming.

Project Objectives

There are several main objectives for this project:

- We will learn new topic i.c. Blue tooth protocol. We will learn how to plan and use resources available to us.
- We will learn how to design and simulate electronic circuits using state of the art tools.
- We will realize the circuit in the form a product fitting into the dimensions.
- We will test the circuit to validate our results against simulated results.
- We will test the circuit for Aero space standards and in the process understand the aero space standards.

- We will compile all the activity into properly designed and worded project report
- We will complete the project in time.

Project Goals

The main goal of this project is to establish a wireless communication between a gyro and a personal computer. To achieve that it is proposed to study about ring laser gyroscope(RLG) and also the way the gyro output is configured in the FPGA for bluetooth communication (i.e. wireless communication) to transfer data from RLG to PC.

Project Scope

This project has a bright scope in Inertial Navigational Systems (INS), Robotics, Controlling a Missile.

CHAPTER II

FUNCTIONING

2.1 Project Setup

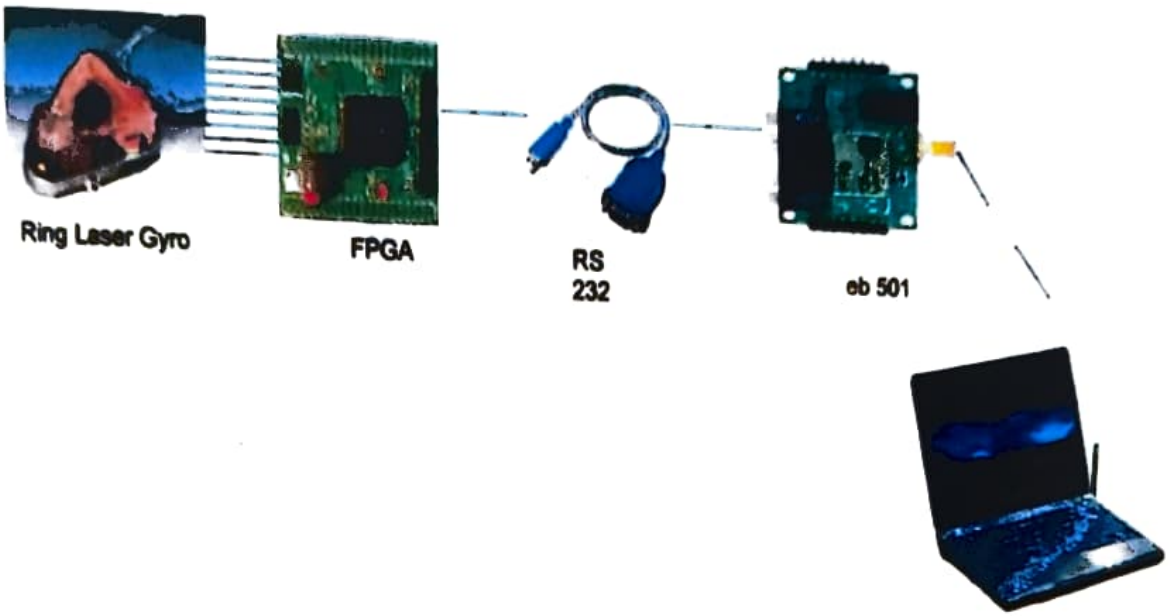


Fig2.1: Project Setup

The above shows the project setup. The output of Ring Laser Gyro is 8-bit parallel data which we need to configure UART in FPGA for serial communication. From RS 232 using extension cord we will transfer data to EB 501 which is bluetooth serial adapter communicates with PC and this operation is vice versa also.

2.2 Ring Laser Gyro

A ring laser gyroscope (RLG) uses interference of laser light within an optical ring to detect changes in orientation and spin. It is an example of a Sagnac interferometer.

The first experimental ring laser gyroscope was demonstrated in the US by Macek and Davis in 1963. The technology has since been developed by a number of companies and establishments world-wide. Many tens of thousands of RLGs are operating in inertial navigation systems and have established high accuracy, with better than 0.01°/hour bias uncertainty, and mean time between failures in excess of 60,000 hours. Ring laser gyroscopes can be used as the stable elements (for one degree of freedom each) in an inertial reference system. The advantage of using an RLG is that there are no moving parts. Compared to the conventional spinning gyroscope, this means there is no friction,which in turn means there will be no inherent drift terms. Additionally, the entire unit is compact, lightweight and virtually indestructible, meaning it can be used in aircraft

2.2.1 WORKING

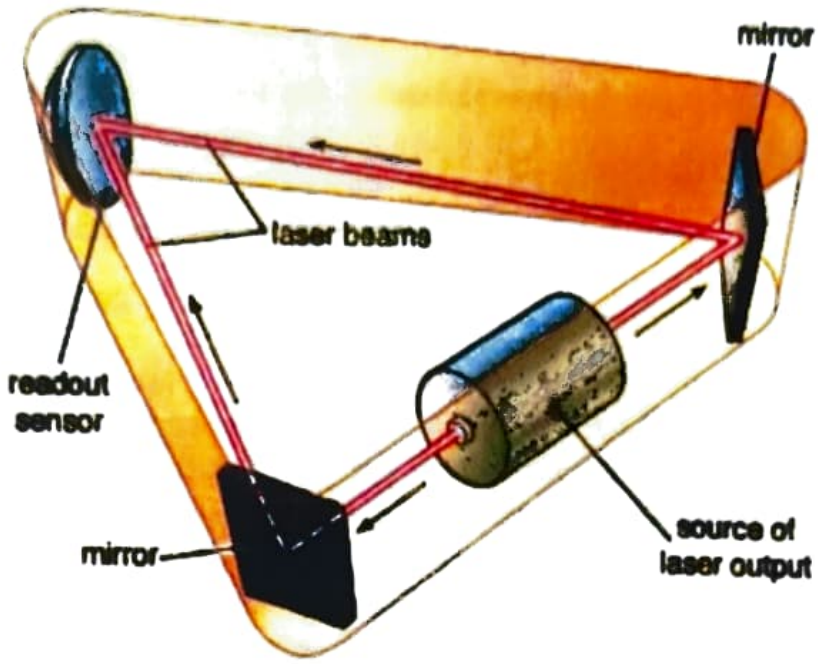


Fig2.2: Showing Ring Laser Gyro Working

RLG's, while more accurate than mechanical gyroscopes, suffer from an effect known as "lock-in" at very slow rotation rates. When the ring laser is rotating very slowly, the frequencies of the counter-rotating lasers become very close (within the laser bandwidth). At this low rotation, the nulls in the standing wave tend to "get stuck" on the mirrors, locking the frequency of each beam to the same value, and the interference fringes no longer move relative to the detector: in this scenario, the device will not accurately track its angular position over time.

NO CALIBRATION

In contrast with the case of passive ring interferometry, in the case of ring laser interferometry there is no need for calibration. With passive ring interferometry there is no way of establishing which position of the interference fringes corresponds to zero anguiar velocity of the ring interferometer setup. Ring laser interferometry, on the other hand, is self calibrating. The beat frequency will be zero if and only if the ring laser setup is non-roting with respect to inertial space.

LOCK IN

Because of the way the laser light is generated, light in laser cavities has a strong tendency to be monochromatic (and usually that is precisely what laser apparatus designers want). This tendency to not split in two frequencies is called lock-in'. The ring laser devices incorporated in navigational instruments (to serve as a ring laser gyroscope) are generally too small to go out of lock spontaneously. By "dithering" the gyro through a small angle at a high audio frequency rate, going out of lock is ensured.

2.2.2 SAGNAC EFFECT

The Sagnac effect (also called Sagnac interference), named after French physicist Georges Sagnac, is a phenomenon encountered in interferometry that is elicited by rotation. The Sagnac effect manifests itself in a setup called ring interferometry. A beam of light is split and the two beams are made to follow a trajectory in opposite directions. To act as a ring the trajectory must enclose an area. On return to the point of entry the light is allowed to exit the apparatus in such a way that an interference pattern is obtained. The position of the obtained. The position of the interference fringes is dependent on the angular velocity of the setup. This arrangement is also called as Sagnac interferometer.

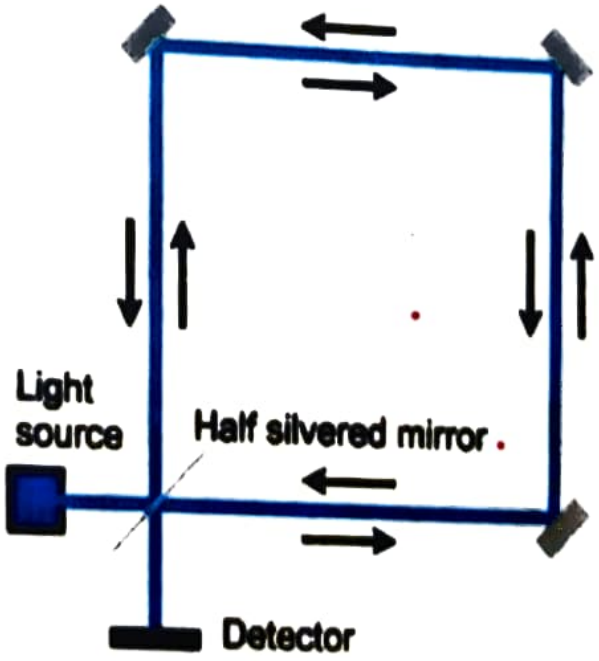


Fig2.3: Schematic representation of a Sagnac interferometer

Usually several mirrors are used, so that the light beams follow a triangular or square trajectory. Fiber optics can also be employed to guide the light. The ring interferometer is located on a platform that can rotate. When the platform is rotating the lines of the interference pattern are displaced as compared to the position of the interference pattern when the platform is not rotating. The amount of displacement is proportional to the angular velocity of the rotating platform. The axis of rotation does not have to be inside the enclosed area.

When the platform is rotating, the point of entry/exit moves during the transit time of the light. So one beam has covered less distance than the other beam. This creates the shift in the interference pattem. Therefore, the interference pattem oblained at each angular velocity of the platform features a different phase-shift particular to that angular velocity.

The Sagnac effect is the electromagnetic counterpart of the mechanics of rotation. A gimbal mounted gyroscope remains pointing in the same direction after spinning up, measures its own angular velocity with respect to the local inertial frame; hence just as a gyroscope it can provide the reference for an inertial guidance system.

2.3 UART

A **universal asynchronous receiver/transmitter** (usually abbreviated **UART** and pronounced) is a type of "asynchronous receiver/transmitter", a piece of computer hardware that translates data between parallel and serial forms. UARTs are commonly used in conjunction with other communication standards such as EIA RS- 232.

A UART is usually an individual (or part of an) integrated circuit used for serial communications over a computer or peripheral device serial port. UARTs are now commonly included in microcontrollers. A dual UART or DUART combines two UARTs into a single chip. Many modern ICs now come with a UART that can also communicate synchronously: these devices are called USARTs.

2.3.1Transmitting and receiving serial data

Serial transmission of digital information (bits) through a single wire or other medium is much more cost effective than parallel transmission through multiple wires. A UART is used to convert the transmitted information between its sequential and parallel form at each end of the link. Each UART contains a shift register which is the fundamental method of conversion between serial and parallel forms.

The UART usually does not directly generate or receive the external signals used between different items of equipment. Typically, separate interface devices are used to convert the logic level signals of the UART to and from the external signaling levels.

External signals may be of many different forms. Examples of standards for voltage signaling are RS-232, RS-422 and RS-485 from the EIA. Historically, the presence or absence of current (in current loops) was used in telegraph circuits.

Some signaling schemes do not use electrical wires. Examples of such are optical fiber, IrDA (infrared), and (wireless) Bluetooth in its Serial Port Profile (SPP). Some signaling schemes use modulation of a carrier signal (with or without wires).

Examples are modulation of audio signals with phone line modems, RF modulation with data radios, and the DC-LIN for power line communication.

Communication may be "full duplex" (both send and receive at the same time) or "half duplex" (devices take turns transmitting and receiving) .

communications. It is useful to communicate between microcontrollers and also with PCs. Many chips provide UART functionality in silicon, and low-cost chips exist to convert logic level signals (such as TTL voltages) to RS-232 level signals (for example, Maxim's MAX232).

2.3.2 Asynchronous receive and transmit

In asynchronous transmitting, teletype-style UARTs send a "start" bit, five to eight data bits, least-significant-bit first, an optional "parity" bit, and then one, one and a half, or two "stop" bits. The start bit is the opposite polarity of the data-line's idle state. The stop bit is the data-line's idle state, and provides a delay before the next character can start. (This is called asynchronous start-stop transmission). In mechanical teletypes, the "stop" bit was often stretched to two bit times to give the mechanism more time to finish printinga character. A stretched "stop" bit also helps resynchronization.

The parity bit can either make the number of "one" bits between any start/stop pair odd, or even, or it can be omitted. Odd parity is more reliable because it assures that there will always be at least one data transition, and this permits many UARTs to resynchronize.

CHAPTER III

Technology

3.1 BLUETOOTH

Bluetooth is a technology, whereby, devices communicate wirelessly to achieve data transfer at the rate of 720kbps within a range of 10 to 100 meters. It operates in the unlicensed ISM (Industrial Scientific and Medical) band at 2.4 gigahertz. The word Bluetooth was borrowed from the 10th century, second King of Denmark. King Harald Bluetooth.

3.1.1 Working of Bluetooth

Bluetooth Special Interest Group manages and maintains the Bluetooth Standard. IEEE has accepted it as 802. 15la standard. Bluetooth was developed with a purpose of creating a single digital wireless protocol, capable of connecting multiple devices and getting over the synchronization issues. It enables short-range wireless communication thus replacing wires connecting the electronic devices.

The Bluetooth RF transceiver lies at physical layer. There are 79 bluetooth channels spaced IMHz apart. A spread spectrum technology is used at the physical layer. Both voice and data transmissions over short distances are possible

A bluetooth device consists of an adapter. A Bluetooth adapter can be built into a device or can be in the form of a card that connects to a device.Instructions are embedded into the device, which enable it to communicate with other devices.

When devices come in each other's radio range, their link managers discover each other. Link management protocol (LMP) engages itself in peer-to-peer message exchange. LMP layer performs link setup and negotiation of packet size.Segmentation and reassembly of packets is done, if needed.

Service delivery protocol enables a bluetooth device to join a piconet. A device inguries what services are available with the piconet. Bluetooth GlobalID is exchanged between the devices. Their profiles are matched and a connection is setup.

Bluetooth uses frequency hopping in timeslots, which means that the bluetooth signals avoid interference with other signals by hopping to a new frequency after transmission or reception of every packet. One packet can cover up to five time slots.

Bluetooth can support an asynchronous data channel, or up to 3 simultaneous synchronous voice channels, or a channel, which concurrently supports asynchronous data and synchronous voice.

Bluetooth technology makes use of the concept of master and slave. Devices have to wait until the master allows them to talk! One master and up to seven slaves employ a star topology to form a piconet.

Piconet

The physical links are created on the basis of masters/slaves (function of the type point to multiple point), a master can control up to seven slaves in his zone.These form a small network called Piconet.

The master is simply the first apparatus connected and is the one that sets the clock. the frequency jumping sequence and the access code for the link. All the Bluetooth modules of the same Piconet use the same frequency jumping sequence and are synchronised with the masters clock.

Whatever happens, the machine's role (master or slave) is invisible to the user.Also, one apparatus can participate in several piconets, being the slave in one and master in another. The interlacing of several piconets forms what is called a Scatternet. Thanks to the frequency jumping, 10 independent piconets (or up to 80 apparatus) can transmit at maximum output.

Above that, the network becomes saturated

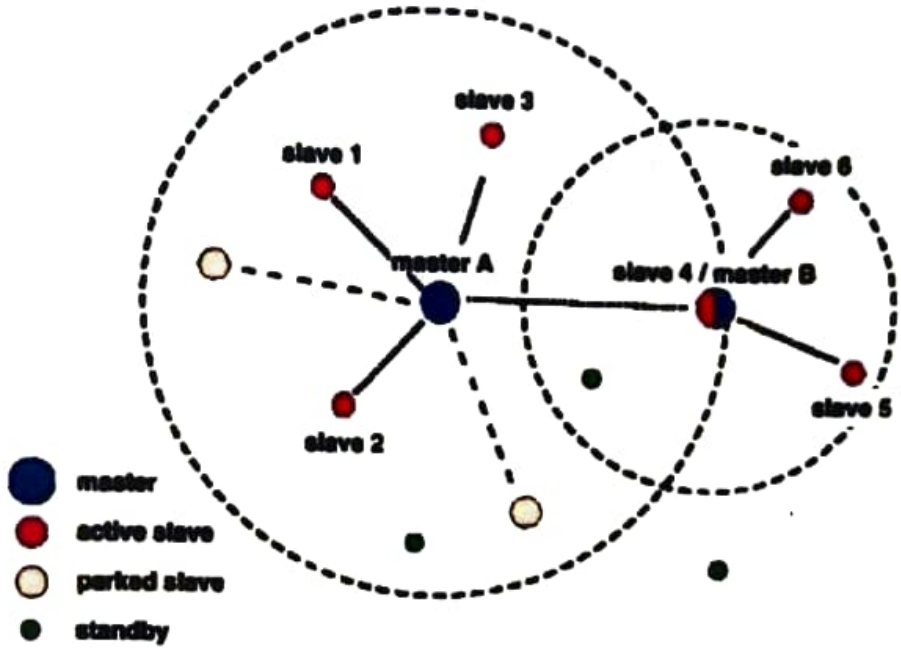


Fig3.1: representing master and slave concept

The transmission diagram of the centre of one piconet is based on the principle of Time Division Duplex (TDD). At first it is the master who sends a packet at a frequency fk), the slave to whom this packet is addressed (and only this one) has the right to reply to it in the time interval following the arrival of the master packet. The reply from the slave is then given on the frequency channel f(k+1). On reception of a master packet, a synchronization word on the top of it enables the slave to re- synchronizes his clock

Bluetooth Protocol Stack

The basic protocols that all Bluetooth systems must have are a radio, a baseband (BB), a link manager (LM), and a logical link controller. The radio takes care of sending and receiving modulated bitstreams. The BB takes care of the timing, and the framing, as well as packets, flow control, error detection, and correction. The LM takes care of managing states and packets and of controlling flow on the link. The logical link controller takes care of multiplexing user protocols, as well as segmentation and Teassembly of larger datagrams into packets, and management. This paper presents the hardware design of a Bluetooth baseband controller which supports an optional high datarate mode of 5 Mbps at a 4 Ms/s symbol rate, as well as a normal data rate of 700 Kbps at I Ms/s

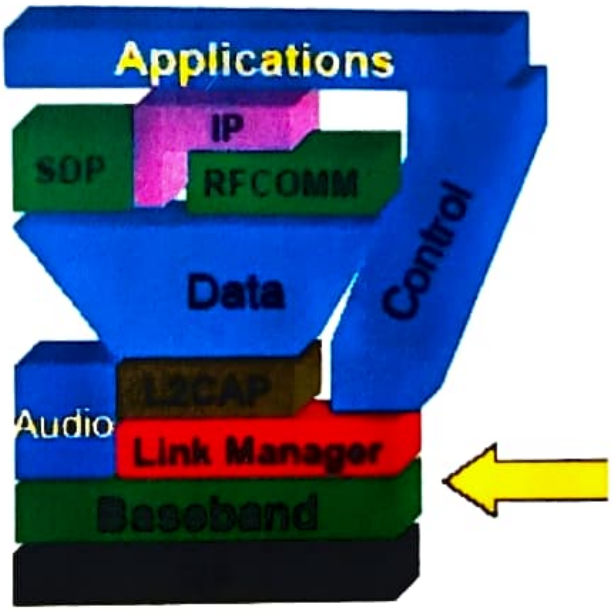


Fig. 3.2. Bluetooth protocol stack.

Bluetooth Application models

File Transfer: -This model talks of an object transfer or transfer of files between devices.

Internet Bridge: -In this model, a cordless modem acts as a modem to a PC and provides dialup networking and faxing.

LAN Access: -Multiple data terminals use a LAN access point (LAP) as a wireless connection to an Ethernet LAN.

Synchronization: -Synchronization model provides a device-to-device synchronization of data.

Headset:-Itis wirelessly connected and can act as an audio input-output interface of remote devices.

3.1.2 Bluetooth's Security System

The Bluetooth protocol allows the authentication and encryption of a link at the same time. The security is controlled by the lower layers of the Bluetooth protocol. A first security level is asured by the fact that each Bluetooth module has a unique address MAC (48-bits). In this way, when the Baracoda scanner connects itself to a terminal with which it is twinned, the bar code is transmitted to this equipment only. To achieve a real level of security. the system, the Bluetooth system necessitates 2 secret keys (authentication key and encryption key) as well as a random number (generated regularly)

Authentication: The scanner sends a challenge to the terminal with which it wishes to connect itself and sends the decoded bar code. The terminal has to reply to the chailenge by returning a link key shared between the scanner and the terminal.

Encryption: When the authentication has been carried out between the 2 devices, the link can be encrypted

3.1.3 Bluetooth Applications

1. Bluetooth has a wide range of applications. Wireless communication between a mobile phone and a handset is possible by way of bluetooth.
2. Computers can form wireless network in a limited space and when bandwidth requirement is less.
3. Unwired communication between the input and output devices of a computer is possible by means of bluetooth technology.

4. Transfer of files and data between devices with OBEX is achieved. OBEX isa session protocol that provides the functionality of HTTP in a lighter fashion.
5. Some game consoles use bluetooth for their wireless controllers.
6. Bluetooth-enabled mobile phones and modems can make possible, dial up Internet connections for PCs or PDAs.

3.1.4 Advantages of Bluetooth

There are numerous advantages of Bluetooth. Firstly, it eliminates all the cords used in connections. Line of sight is not required. The word 'unwired' is in some way synonymous to the word 'uncluttered'.

Secondly, with bluetooth headsets, you can use your cell phone without the use of your hands. That makes it safe to talk on phone while your hands are engaged in other activities. Due to bluetooth, you are not required to be physically close to the device you are using.

Thirdly, Bluetooth devices are fairly inexpensive. There is no special cost incurred in using the service.

The next concern is interoperability. Bluetooth is a standardized specification. Bluetooth enabled devices are highly compatible. They understand each other without human intervention. When they enter a range of one another, they start communicating on their own. The process of setup is automatic.

Then comes efficiency! Bluetooth uses low power signals, thus requiring less energy. Due to spread-spectrum frequency hopping, interference with other wireless devices stays away.

Bluetooth Special Interest Group has been working on upgraded versions, which can yet remain backward compatible.

Moreover, bluetooth is secure. Strict security rules will not allow the devices to communicate unless pre-approved by you. So it is you, who is the controller of communication.

Ericsson was the first to develop the Bluetooth specification. Many companies are now making their products capable to use it. I think bluetooth technology is a complete package of virtues like feature simplicity, inexpensiveness and the need of least human involvement in its implementation. Bluetooth is here to stay for years to come.

3.2 FPGA

The historical roots of FPGAs are in complex programmable logic devices (CPLDs) of the early to mid 1980s. A Xilinx co-founder invented the field programmable gate array in 1984. CPLDs and FPGAs include a relatively large number of programmable logic elements. CPLD logic gate densities range from the equivalent of several thousand to tens of thousands of logic gates, while FPGAs typically range from tens of thousands to several million.

The primary differences between CPLDs and FPGAs are architectural. A CPLD has a somewhat restrictive structure consisting of one or more programmable sum-of-products logic arrays feeding a relatively small number of clocked registers.The result of this is less flexibility, with the advantage of more predictable timing delays and a higher logic-tointerconnect ratio. The FPGA architectures, on the other hand, are dominated by interconnect. This makes them far more flexible but also far more complex to design for.

Another notable difference between LDs and FPGAs is the presence in most FPGAs of higher-level embedded functions (such as adders and multipliers) and embedded memories.

Some FPGAs have the capability of partial re-configuration that lets one portion of the device be re-programmed while other portions continue running.

3.2.1 ARCHITECTURE

The typical basic architecture consists of an array of configurable logic blocks (CLBs) and routing channels. Multiple VO pads may fit into the height of one row or the wIdth of one column in the array. Generally, all the routing channels have the same width.

An application circuit must be mapped into an FPGA with adequate resources.

A classic FPGA logic block consists of a 4-input lookup table (LUT), and a flip-flop. as shown below. In recent years, manufacturers have started moving to 6- input LUTs in their gn performance parts, claiming increased performance.

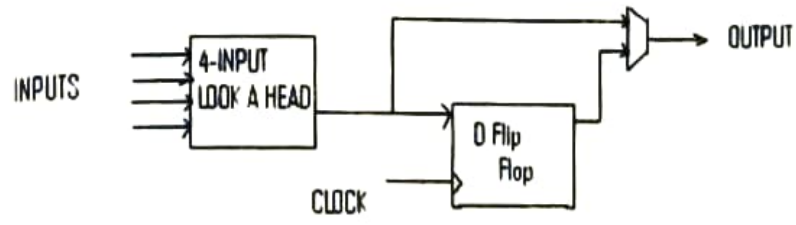


Fig3.3: TYPICAL LOGIC BLOCK

There is only one output, which can be either the registered or the unregistered LUT ourput. The logic block has four inputs for the LUT and a clock input. Since clock signals (and often other high-fanout signals) are normally routed via special-purpose dedicated routing networks in commercial FPGAs, they and other signals are separately managed.

3.2.2 MORDERN DEVELOPMENTS

Inclusion of abundant memory recourses in FPGAs has made them suitable for implementation of embedded system on programmable chip (SoPC), where a complete system fits on a single programmable chip with the processor unit also embedded inside the FPGA. Two different approaches to include processor cores in an FPGA are: hard-processor core, soft-processor core

HARD PROCESSOR CORE

A hard processor core has dedicated silicon on the FPGA. This allows it to operate with a core frequency and have a DMIPS (Dhrystone million of instructions per second) rating similar to that of a discrete micro processor.

A benefit of the hard core is that it exists in an environment where the surrounding peripherals can be customized for the application. Unfortunately, it does not provide the ability to adjust the core for the application nor does it allow for the flexibility of adding a processor to an existing design or an additional processor for more processor capabilities.

SOFT PROCESSOR CORE

A soft core processor solution is implemented entirely in the logic primitives of an FPGA since soft processor cores are provided as synthesizable hardware descriptive language (HDL), they inherently provide more design flexibility than hard cores as the designer can modify the core interface to fit better into a specific design.

Soft cores provide even greater flexibility by being configurable. A configurable core may include a graphical tool that enables certain functions to be included or excluded with out having to manually modify HDL source code

3.2.3 FPGA DESIGN AND PROGRAMMING:

To define the behavior of the FPGA the user provides a hardware description language (HDL) or a schematic design. Common HDLs are VHDL and Verilog. Then. using an electronic design automation tool, a technology-mapped netlist is generated. The netlist can then be fitted to the actual FPGA architecture using a process called place-and-route, usually performed by the FPGA Company's proprietary place-and-route software.

The user will validate the map. place and route results via timing analysis, simulation, and other verification methodologies. Once the design and validation process is complete, the binary file generated (also using the FPGA company's proprietary software) is used to (reconfigure the FPGA.

In an attempt to reduce the complexity of designing in HDLs, which have been compared to tne equivalent of assembly languages, there are moves to raise the abstraction level of the design.. Approaches based on standard C or C++ (with libraries or other extensions allowing parallel programming) are found in the Catapult C tools from Mentor Graphics, and in the Impulse C tools from Impulse Accelerated Technologies. Annapolis Micro Systems, Inc.'s Core Fire Design Suite and National Instruments Lab VIEW FPGA provide a graphical dataflow approach to high-level design entry. Languages such as System Verilog.

System VHDL, and Handel-C (from Celoxica) seek to accomplish the same goal, but are aimed at making existing hardware engineers more productive versus making FPGAs more accessible to existing software engineers. There is more information on C to HDL and Flow to HDL in their respective articles.

To simplify the design of complex systems in FPGAs, there exist libraries of predefined complex functions and circuits that have been tested and optimized to speed up the design process. These predefined circuits are commonly called IP cores, and are available from FPGA vendors and third-party IP suppliers (rarely free, and typically released under proprietary licenses). Other predefined circuits are available from developer communities such as Open Cores (typically free, and released under the GPL, BSD or similar license), and other sources.

In a typical design flow, an FPGA application developer will simulate the design at multiple stages throughout the design process. Initially the RTL description in VHDL or Verilogis simulated by creating test benches to simulate the system and observe results. Then, after the synthesis engine has mapped the design to a netlist, the netlist is translated to a gate level description where simulation is repeated to confirm the synthesis proceeded without errors. Finally the design is laid out in the FPGA at which point propagation delays can be added and the simulation un again with these values back-annotated onto the netlist.

3.2.4 APPLICATIONS

Applications of FPGAs include digital signal processing, software-defined radio, aerospace and defence systems, ASIC prototyping, medical imaging. computer vision, speech recognition, cryptography, bioinformatics, computer hardware emulation and a growing range of other areas.

FPGAs especially find applications in any area or algorithm that can make use of the massive parallelism offered by their architecture. One such area is code breaking, in particular brute, force attack, of cryptographic algorithms.

FPGAS are increasingly used in conventional High Performance Computing applications where computational kernels such as FFT or Convolution are performed on the FPGA instead of a microprocessor. The use of FPGAs for computing tasks is known as reconfigurable computing.

The inherent parallelism of the logic resources on the FPGA allows for considerable compute throughput even at a sub-500 MHz clock rate. For example, the current (2007) generation of FPGAs can implement around 100 single precision floating point units, all of which can compute a result every single clock cycle. The flexibility of the FPGA allows for even higher performance by trading off precision and range in the number format for an increased number of parallel arithmetic units. This has driven a new type of processing called reconfigurable computing, where time intensive tasks are offloaded from software to FPGAs.

CHAPTER IV

Hardware and Software Implementation

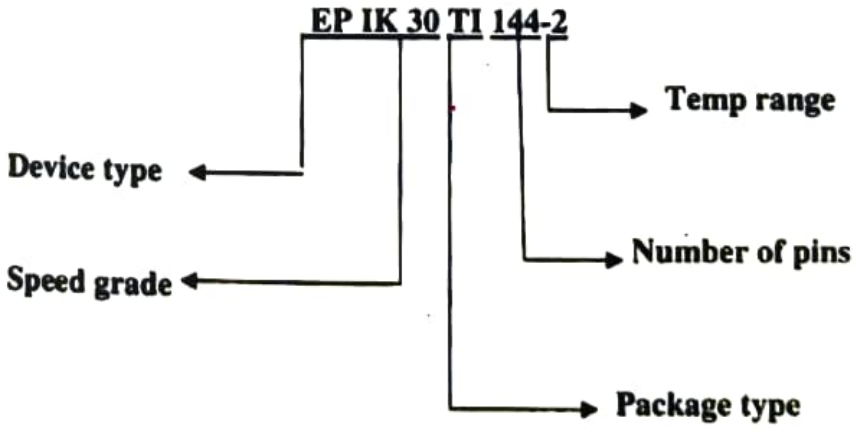
4.1 FPGA (EP IK 30 TI 144-2)

The FPGA used in this project is ACEXIK (EPI K30TI144-2).

Altera ACEX IK devices range in density from 576 to 4,992 logic elements (Les). They devices provide a die-efficient, low-cost architecture by combining look-up table (LUT) architecture with EABs. LUT-based logic provides optimized performance and efficiency for data-path, register intensive, mathematical, or digital signal processing (DSP) designs. The devices feature high-performance embedded RAM blocks that can implement dual-port RAM, read-only memory (ROM), first-in first-out buffers (FIFO) or logic. ACEX IK devices also feature Multi Volt TM VO operation, enabling them to interface with devices running at 5.0 V, 3.3 V, and 2.5 V.

4.1.1 Ordering information

The model of FPGA ACEX 1K used is EPIK30TI144-2A



4.1.2 Features

- Programmable logic devices (PLDs). providing low cost system-on-a programmablechip (SOPC) integration in a single device.
- Enhanced embedded array for implementing mega functions such as efficient memory and specialized logic functions.
- Dual-port capability with up to 16-bit width per embedded array block (EAB). Logic array for general logic functions
- High density

  4.0.0 10.000 to 100,000 typical gates (see Table 1)

  4.0.0 Up to 49,152 RAM bits

- Cost-efficient programmable architecture for high-volume applications

  4.0.0 Cost-optimized process

  4.0.0 Low cost solution for high-performance communications applications

- System-level features

  4.0.0 Multi Volt TM VO pins can drive or be driven by 2.5-V, 3.3-V or 5.0-V devices

  4.0.0 Low power consumption

  4.0.0 Bidirectional 10 performance up to 250MHz

  4.0.0 Fully compliant with the peripheral component interconnect

- Extended temperature range
- Flexible interconnect

  - Interconnect continuous routing structure for fast, predictable interconnect delays

  4.0.0 Dedicated carry chain that implements arithmetic functions such as fast adders, counters, and comparators.

  4.0.0 Dedicated cascade chain that implements high-speed, high-fan-in logic functions.

  4.0.0 Tri-state emulation that implements internal tri-state buses

- Powerful I/O pins

4.0.0Individual tri-state output enable control for each pin

4.0.0 Open-drain option on each /O pin

4.0.0 Programmable output slew-rate control to reduce switching noise

Clamp to Vcc IO user-selectable on a pin-by-pin basis- Supports hot-socketing

4.1.3 Architecture

- Logic Array Block

An LAB consists of eight Les, their associated carry and cascade chains, LAB control signals, and the LAB local interconnect. The LAB provides the coarse-grained structure to the ACEX IK architecture, facilitating efficient routing with optimum device utilization and high performance.

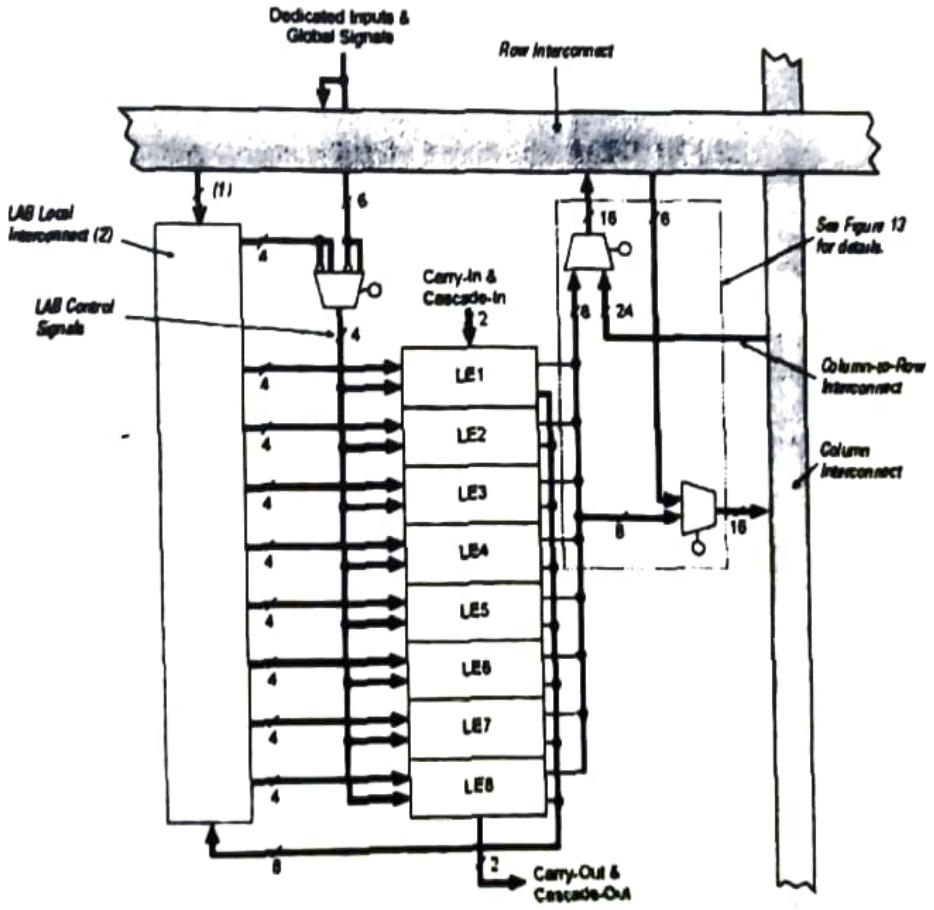


Fig4.1: Logic array blocks of FPGA chip

Each LAB provides four control signals with programmable inversion that can be Used in all eight LEs. Two of these signals can be used as clocks, the other two can be used as clear/preset control. The LAB clocks can be driven by the dedicated clock input pin, global signals, I/O signals, or internal signals via the LAB local interconnect. The LAB preset and clear control signals can be driven by the global signals, I/O signals, or internal signals via the LAB local interconnect. The global control signals are typically used for global clock, clear, or preset signals because they provide asynchronous control with very low skew across the device. If logic is required on a control signal, it can be generated in one or more Les in any LAB and driven into the local interconnect of the target LAB .

Logic Element:

The LE, the smallest unit of logic in the ACEX 1K architecture, has a compact size that provides efficient logic utilization. Each L.E contains a 4-input LUT, which is a function generator that can quickly compute any function of four variables. In addition, each LE contains a programmable flipflop with a synchronous clock enable, a carry chain, and a cascade chain. Each LE drives both the local and the Fast Track Interconnect routing structure. Figure 8 shows the ACEX IK LE.

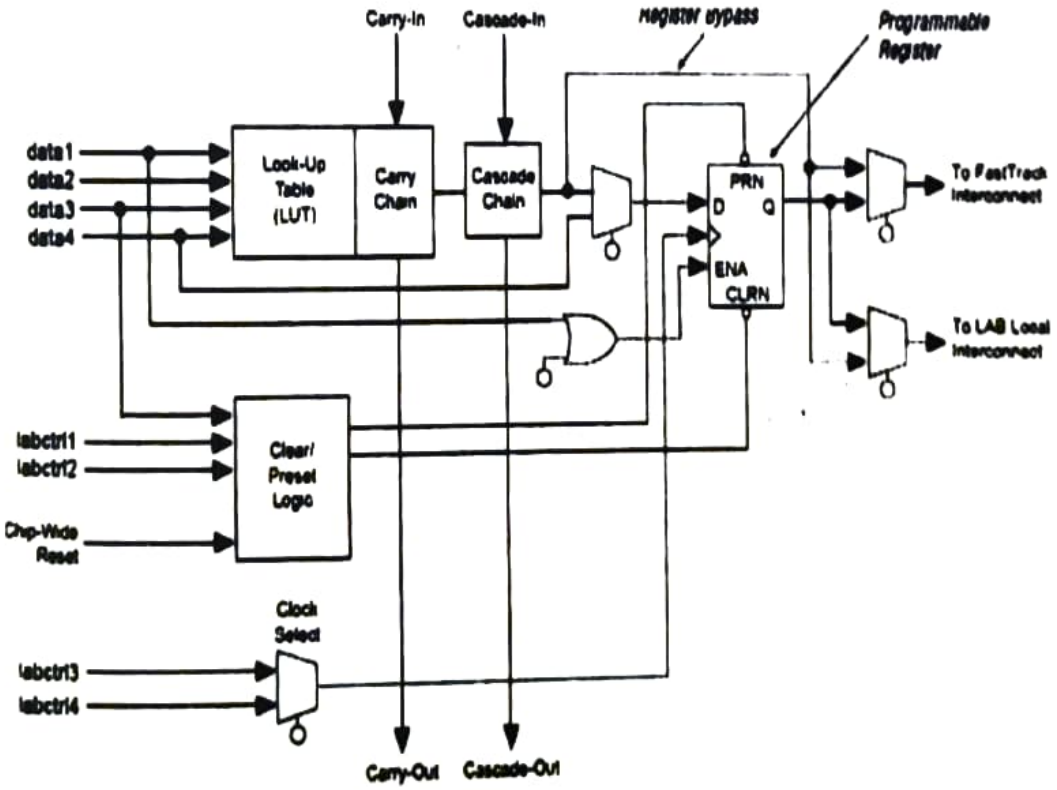


Fig 4.2. Elementsof FPGA chip

The programmable flipflop in the LE can be configured for D. T., JK, or SR operation. The clock, clear and preset control signals on the flipflop can be driven by global signals. general-purpose I/0 pins, or any internal logic. For combinatorial functions the flipflop is bypassed and the LUT's output drives the LE's output.

The LE has two outputs that drive the interconnect, one drives the local Interconnect, and the other drives either the row or column Fast Track Interconnect routing structure. The two outputs can be controlled independently. For example, the LUT can drive one output while the register drives the other output. This feature, called register packing, can improve LE utilization because the register and the LUT can be used for unrelated functions.

The ACEX IK architecture provides two types of dedicated high-speed data paths that connect adjacent LEs without using local interconnect paths: carry chains and cascade chains. The carry chain supports high-speed counters and adders, and the cascade chain implements wide-input functions with minimum delay. Carry and cascade chains connect all Les in a LAB and all LABs in the same row. Intensive use of carry and cascade chains can reduce routing flexibility. Therefore, the use of these chains should be limited to speed-critical portions of a design.

- **Carry Chain**

The carry chain provides a very fast (as low as 0.2 ns) carry-forward function between LEs. The carry-in signal from a lower-order bit drives forward into the higher-order bit via the carry chain, and feeds into both the LUT and the next portion of the carry chain. This feature allows the ACEX IK architecture to efficiently implement high-speed counters, adders, and comparators of arbitrary width. Carry chain logic can be created automatically by the compiler during design processing, or manually by the designer during design entry.Parameterized functions, such as LPM and Design Ware functions, automatically take advantage of carry chains.

- **Cascade Chain**

With the cascade chain, the ACEX IK architecture can implement functions that have a very wide fan-in. Adjacent LUTs can be used to compute portions of the function in parallel; the cascade chain serially connects the intermediate values. The cascade chain can use a logical AND or logical OR (via De Morgan's inversion) to connect outputs of adjacent LEs. With a delay as low as 0.6 ns per LE, each additional LE provides four more inputs to the effective width of a function. Cascade chain logic can be created automatically by the compiler during design processing. or manually by the designer during design entry.

4.1.4 Functional Description

Each ACEX IK device contains an enhanced embedded array that implements memory and specialized logic functions, and a logic array that implements general logic. The embedded array consists of a series of EABs. When implementing memory functions, each EAB provides 4,096 bits, which can be used to create RAM, ROM, dual-port RAM, or first in first-out (FIFO) functions. When implementing logic, each EAB can contribute 100 to 600 gates towards complex logic functions such as multipliers, microcontrollers, state machines, and DSP functions. EABs can be used independently, or multiple EABs can be combined to implement larger functions.

The logic array consists of logic array blocks (LABs). Each LAB contains eight LEs and a local interconnect. An LE consists of a 4-input LUT, a programmable flipflop. and dedicated signal paths for carry and cascade functions. The eight LEs can be used to create medium-sized blocks of logic-such as 8-bit counters, address decoders, or state machines or combined across LABs to create larger logic blocks. Each LAB represents about 96 usable logic gates.

Signal interconnections within ACEX 1K devices (as well as to and from device pins) are provided by the FastTrack Interconnect routing structure, which is a series of fast, continuous row and column channels that run the entire length and width of the device.

Each I/O pin is fed by an I/O element (1OE) located at the end of each row and column of the FastTrack Interconnect routing structure. Each IOE contains a bidirectional VO buffer and a flipflop that can be used as either an output or input register to feed input, output, or bidirectional signals. When used with a dedicated clock pin. these registers provide exceptional performance.As inputs, they provide setup times as low as 1.1 ns and hold times Or O ns. As outputs, these registers provide clock-to-output times as low as 2.5 ns. IOEs provide a variety of features, such as JTAG BST support, slew-rate control, tri-state buffers, and open-drain outputs.

I/O BLOCK

ACEX IK devices provide six dedicated inputs that drive the flip-flops' control fnnuts and ensure the efficient distribution of high-speed, low skew (Less than 1.0 ns) control signals. These signals use dedicated routing channels that provide shorter delays and lower skews than the FastTrack Interconnect routing structure. Four of the dedicated inputs drive four global signals. These four global signals can also be driven by internal logic, providing an ideal solution for a clock divider or an internally generated asynchronous clear signal that clears many registers in the device.

Figure 3 shows a block diagram of the ACEX 1K device architecture. Each group of LEs is combined into an LAB: groups of LABs are arranged into rows and columns. Each row also contains a single EAB. The LABs and EABs are interconnected by the FastTrack Interconnect routing structure.IOEs are located at the end of each row and column of the FastTrack Interconnect routing structure.

Embedded Array Block (EAB)

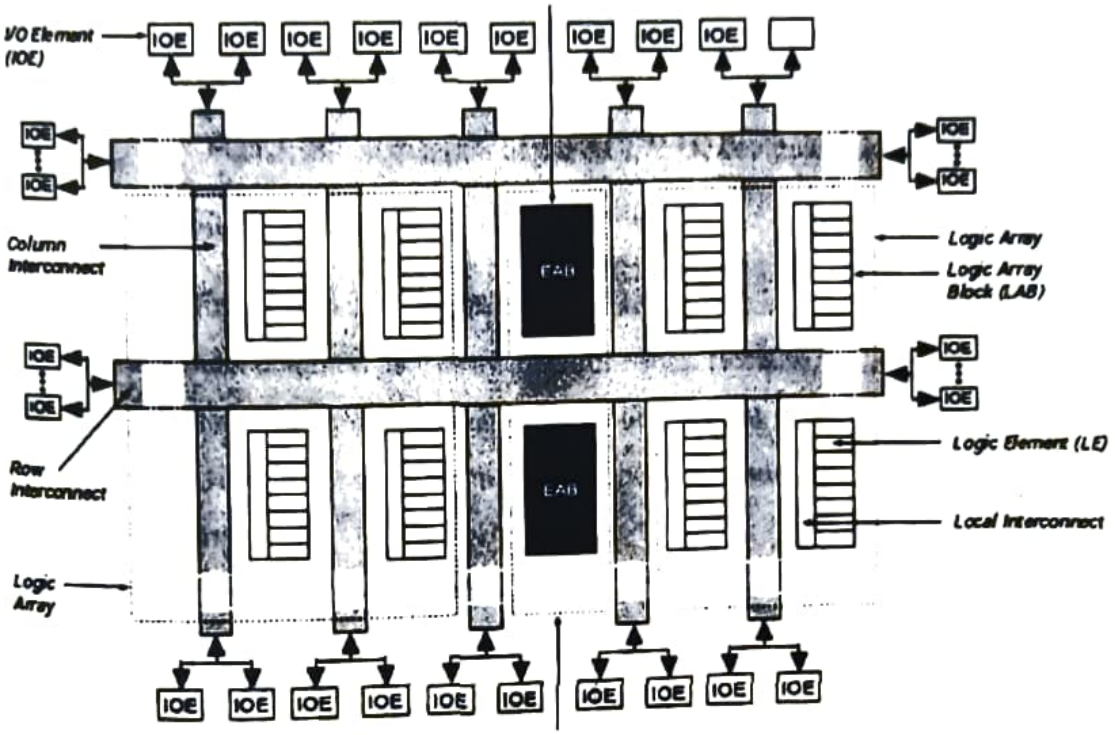


Fig 4.3: Block diagram of the ACEX 1 K device architecture

Interconnect routing structure. IOEs are located at the end of row and column of the Fast Track Interconnect routing structure.

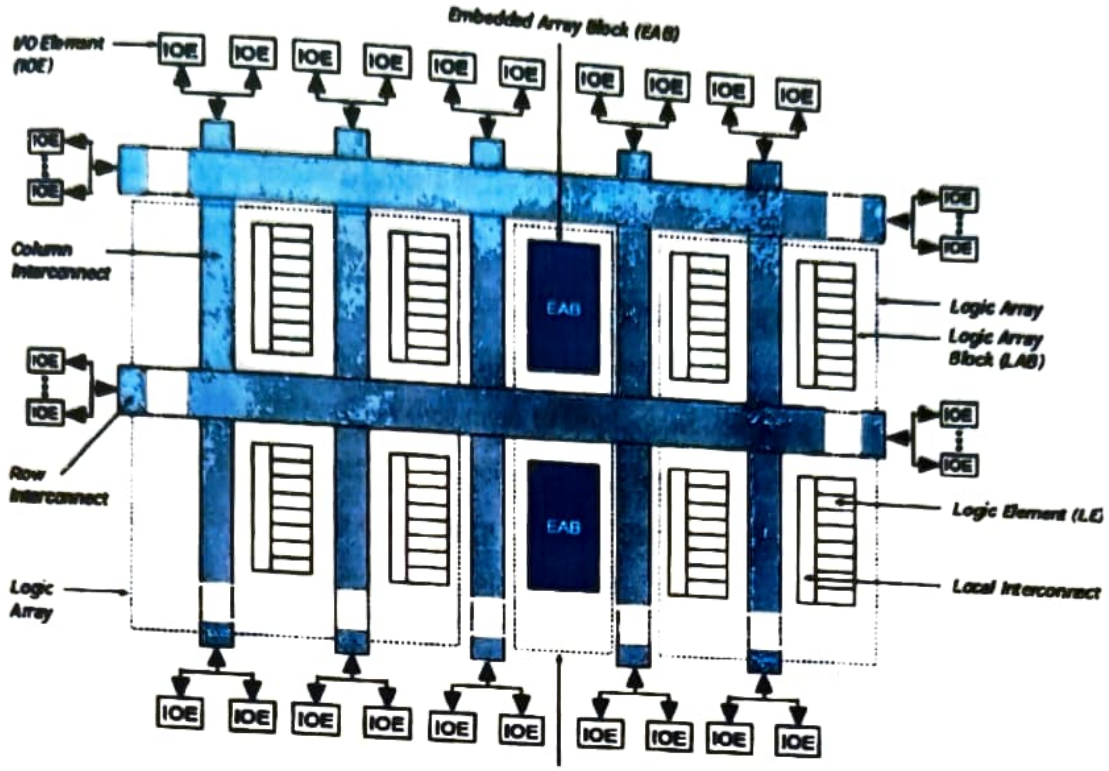


Fig4.3: Block diagram of the ACEX I K device architecture

Embedded Array Block

The EAB is a flexible block of RAM, with registers on the input and output ports, that is used to implement common gate array mega functions. Because it is large and flexible,the EAB is suitable for functions such as multipliers, vector scalars, and error correction Circuits. These functions can be combined in applications such as digital filters and microcontrollers.

Logic functions are implemented by programming the EAB with a read only pattern during configuration, thereby creating a large LUT. With LUTs, combinatorial functions are implemented by looking up the results rather than by computing them. This implementation of combinatorial functions can be faster than using algorithms implemented in general logic, a performance advantage that is further enhanced by the fast access times of EABs The capacity of EABs enables designers to implement complex functions in a single logic level without the routing delays associated with linked LEs or field-programmable gate array (FPGA) RAM blocks. For example, a single EAB can implement any function with 8 inputs and 16 outputs. Parameterized functions, such as LPM functions can take advantage of the EAB automatically.

RAM

The ACEX IK enhanced EAB supports dual-port RAM. The dual-port structure is ideal for FIFO buffers with one or two clocks. The ACEX IK EAB can also support up to 16-bit-wide RAM blocks. The ACEX IK EAB can act in dual-port or single-port mode. When in dual port mode, separate clocks may be used for EAB read and write sections, allowing the EAB to be written and read at different rates. The EAB can also be used for bidirectional, dual-port memory applications where two ports read or write simultaneously. To implement this typeof dual-port memory, two EABs are used to support two simultaneous reads or writes. Alternatively, one clock and clock enable can be used to control the input registers of the EAB,while a different clock and clock enable control the output register

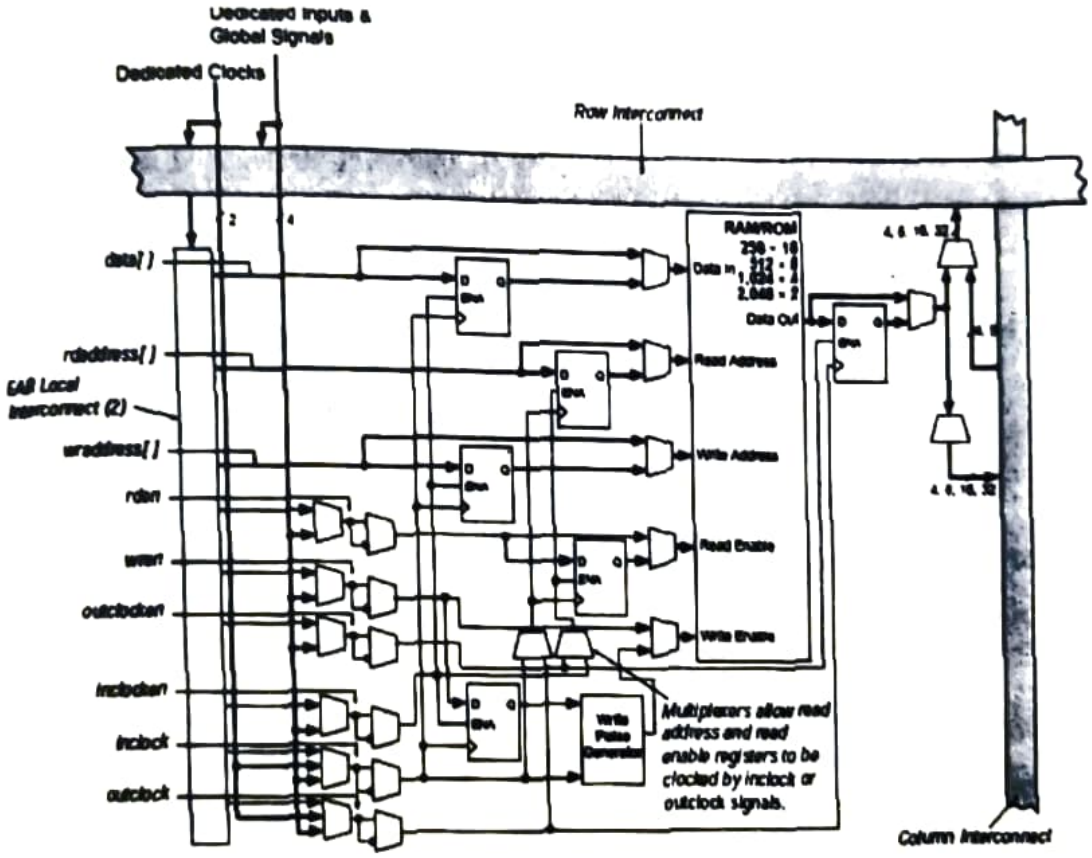


Fig4.4: ACEX I K Device in Dual-Port RAM Mode

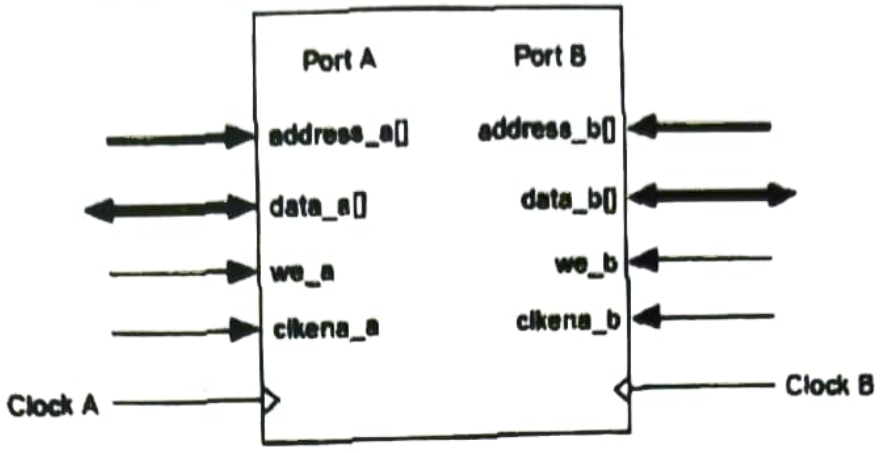


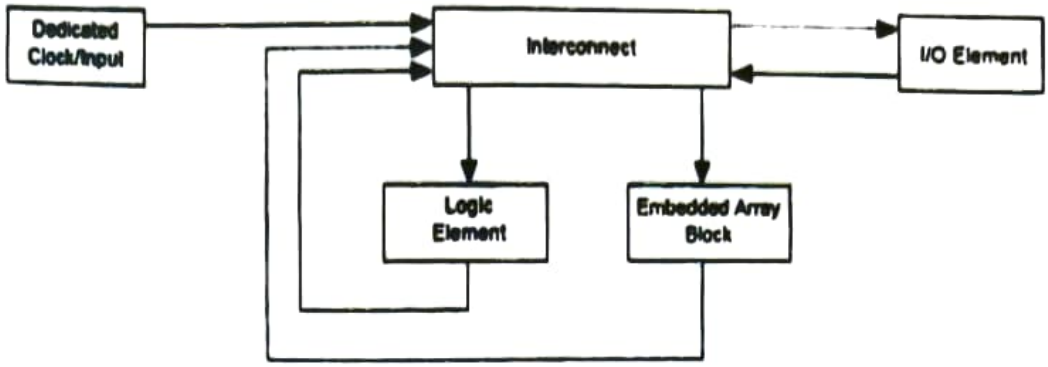
Fig4.5: ACEX I K Device in single port RAM MODE

4.1.5 Timing Model

The continuous, high-performance FastTrack Interconnect routing resources Ensure accurate simulation and timing analysis as well as predictable performance.This predictable performance contrasts with that of FPGAs, which use a segmented connection scheme and, therefore, have an unpredictable performance. Device performance can be estimated by following the signal path from a source, through the interconnect, to the destination. For example ,the registered performance between two Les on the same row can be calculated by adding the following parameters.

- LE register clock-to-output delay(tCO)
- Interconnect delay (tSAMEROW)
- LE look-up table delay (tLUT)
- LE register setup time(tSU)

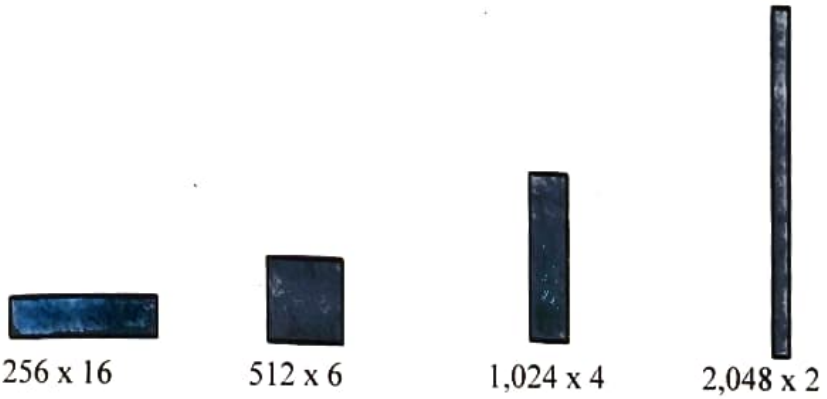
The routing delay depends on the placement of the source and destination LEs. A more complex registered path may involve multiple combinatorial LEs between the source and destination LEs. Timing simulation and delay prediction are available with the simulator and Timing Analyzer. or with industry-standard EDA tools. The Simulator offers both pre-synthesis functional simulation to evaluate logic design accuracy and post- synthesis timing simulation with 0.1-ns resolution. The Timing Analyzer provides point- to-point timing delay information, setup and hold time analysis, and device-wide performance.



F1g4.6: ACEX lK Device Timing Model

EABs can be used to implement synchronous RAM, which is easier to use than asynchronous RAM.A circuit using asynchronous RAM must generate the RAM write enable signal, while ensuring that its data and address signals meet setup and hold time specifications relative to the write enable signal .In contrast ,the EAB's synchronous RAM generates its own write enable signal and is self-timed with respect to the input or write clock. A circuit using the EAB's self-timed RAM must only meet the setup and hold time specifications of the global clock. When used as RAM ,each EAB can be configured in any of the following sizes.256x16; 512x8; 1024x4; or 2048x2.

ACEX 1K EAB Memory Configurations



Examples of Combining ACEX IK EABS

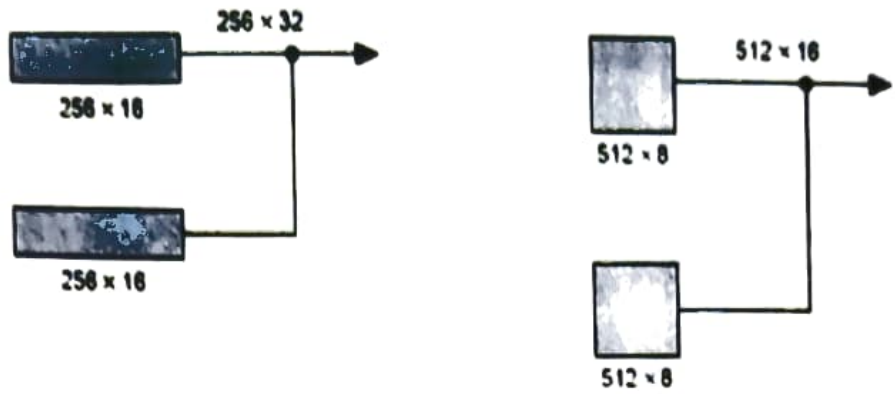


Fig4.7: ACEX IK EAB memory configuration

If necessary, all EABs in a device can be cascaded to form a single RAM block. EABs can be Cascaded to form RAM blocks of up to 2,048 words without impacting timing. Altera Software automatically combines EABs to meet a designer's RAM specifications.

4.2 Bluetooth serial Adaptor- eb501

The eb501 Bluetooth Serial adapter makes use of the eb101 Bluetooth Serial Module with RF pad. The antenna connection is made via a SMA connector. Unlike the eb301 designs the eb50l provides various interfaces one single adapter: specifically providing a RS232 level interface, a SV logic interface and a 3.3V logic interface. The complete design with package for the eb501 is available on the A7 website including gerbers , schematic, and BOM.

The eb501 is powered via a separate DC input or may be powered through the respective interface if using 5V or 3.3V logic. If powering through J4, 5 to 14 volts may be supplied. The following picture identifies many of the features of the ebs01 adapter as well as dimensions.

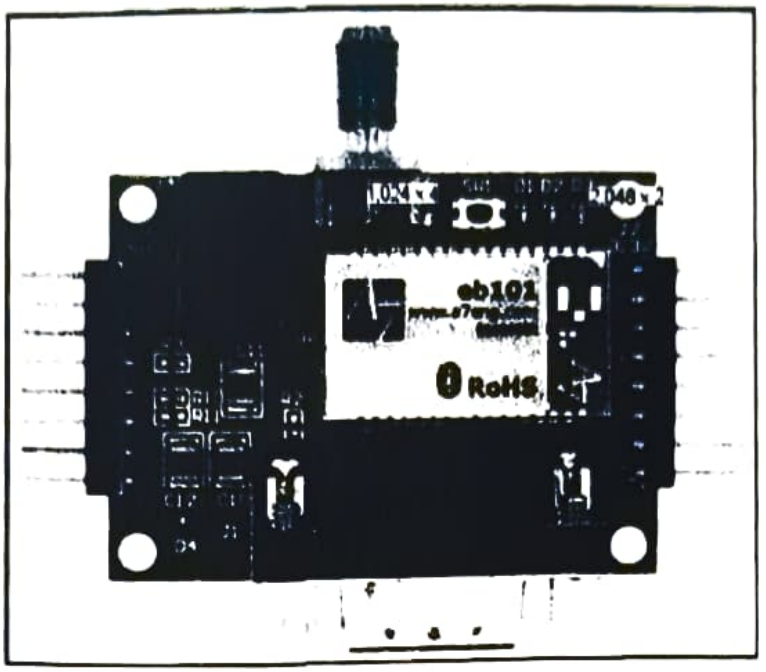


Fig 4.8: Bluetooth serial Adaptor eb501

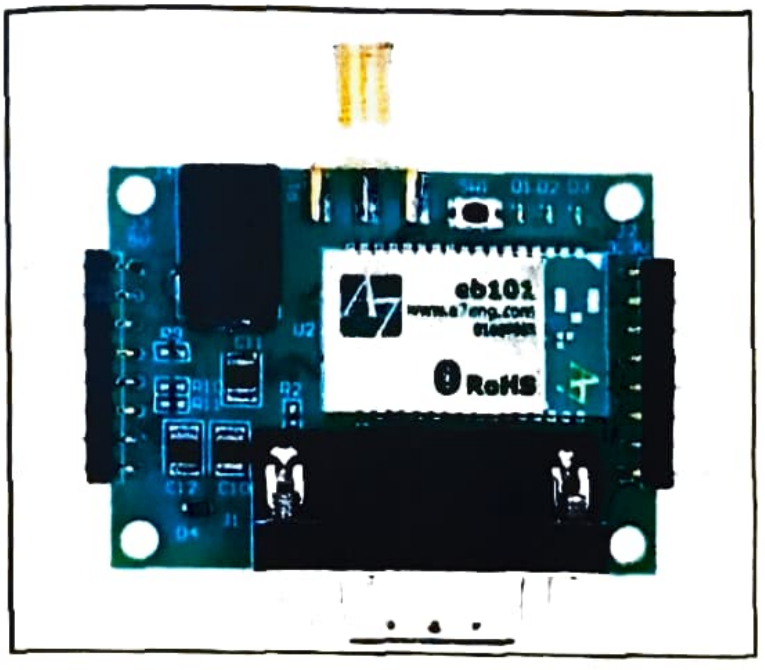


Fig 4.8: Bluetooth serial Adaptor- eb501

4.2.1 eb101: Design Features

- No external components required
- Simple serial UART communications
- Easy Connect enabled
- 2.4GHz FHSS technology ensures high reliability and robustness to interference
- Unique surface mount pad design that can be flow soldered or hand soldered
- Low current consumption
- Small footprint 14.5mm x 25mm x 2.1 mm
- Up to 320kbps continuous throughput
- Fully Qualified Bluetooth v2.0+EDR
- FCC Part 15 Modular Approval
- RoHS certified
- Supports 2.2- 4.2V supply voltage
- Supports 1.8 to 3.6V I/0.

General Description

The eb101 surface mount module is a highly integrated and easy to use Bluetooth solution designed for low cost and reliability. The eb101 implements all components of the Bluetooth stack on board so that additional host processor code is not required. Once a connection to another Bluetooth device has been established, the link has the appearance of a cabled serial connection eliminating the need for special wireless protocol knowledge. Simple UART communication facilitates the interface between the host processor and the eb101 radio. This UART interface may be used to discover, connect, and communicate with other Bluetooth devices through a simple ASCII command set. Easy Connect provides the option for a drop-in cable replacement solution without having to issue commands. A simple push button and LED provide all the control necessary to establish and maintain a wireless connection.

Applications

- Medical Equipment
- POS Systems
- Telemetry Systems
- Industrial Automation

- Barcode and RFID scanners
- Lighting Control
- Robotics

The eb101 surface mount module is ideal for enabling cost sensitive designs with a widely supported industry standard wireless protocol. Monitoring and control applications will benefit from an integrated implementation of the serial port profile for seamless connectivity with desktop computers, PDAs, and cellular phones. A focus on low current consumption makes the eb101 ideal for use in standalone battery powered devices common to medical and remote data capture applications. The eb101 has been designed to simplify both hardware and software integration of Bluetooth to reduce development costs. The highly integrated design and surface mount package streamlines production in both low and high volume applications.

4.2.2 Key Features

Radio

- Based on CSROs BC4-Ext Bluetooth Chip.
- +6dBm RF transmit power w/level control.
- Bluetooth v2.0+EDR compliant.
- Integrated Antenna, U.FL connector, and RF pad versions available.
- Support for Adaptive Frequency Hopping (AFH) and 802.11 coexistence.
- 2.4GHz FHSS technology ensures high reliability and is robust to interference.
- Full reference designs are available.

Firmware

- Simple ASCII interface for command and control of Bluetooth technology
- Easy Connect enabled for simple cable replacement solution
- Secure point to point communications .
- 56-bit encryption . Fully embedded Bluetooth stack including the Discovery Profile (SDP). Dial Up Networking Profile (DUN) and Serial Port Profile energic Access Profile (GAP), Service (SPP)
- Wireless connection status output line
- Host wakeup output line (future firmware function)
- Break control input line

UART Interface

- Baud rates from 1.2kbps to 460.8kbps are supported
- Optional hardware flow control
- Standard 8 bit, no parity, one stop bit (8NI) communications 1.8-3.6V logic levels for Tx, Rx, and I/O lines

Power

- 2.2-4.2VDC supply voltage
- Low operating power consumption
- RESET line controls module boot and restart

Device

- Up to 320kbps continuous data transfer rate
- Unique surface mount pad design that can be flow soldered or hand soldered
- Small footprint 14.5mm x 25mm x 2.1mm
- All devices have a globally unique ID

Data Integrity

- CQDDR increases the effective data rate in noisy environments.
- RSSI used to minimize interference to other radio devices using the ISM band.

4.2.3 Device Terminal Descriptions Radio and Control

- RESET

When this line is driven low, the eb101 module will enter reset mode and remain there until this line is driven high. This line is debounced so it must be held low for> 5ms to cause a reset. Use of this line is optional.

- STATUS

This is an output line that can be used to monitor the status of a Bluetooth connection in both command mode and Easy Connect mode. This line will be low when there is an active connection and high when there is no connection. A maximum of 8mA of current may be drawn from this line. Use of this line is optional.

- BREAK

Drive this line low for >5ms to switch to the command channel so that commands can be issued by the host over the UART connection. This line is equivalent to the soft break command. Use of this line is optional.

- SWITCH

Drive this line low for >10 seconds on power up to initiate a reset to the factory default settings. Drive the line low for >5 seconds at any time after power up to initiate Easy Connect. The application circuit in figure 7.I shows proper usage of this line with a momentary switch that pulls the line to ground when pressed. Use of this line is optional.

- IND LED

This is an output line that is designed to be connected to a small LED for a visual indication of the current operating mode. Use of this line is standard when supporting Easy Connect but it is useful in command mode as well. The example application circuit in figure 7.1 shows proper usage. A maximum of 8mA of current may be drawn from this line to drive an LED. Use of this line is optional.

- WAKEUP (Future Firmware Function)

Drive this line low to keep the module awake when sleep mode is enabled. The module will be awake within 5ms after this line is driven low. Use of this line is optional.

- HOST_WAKEUP (Future Firmware Function)
  When enabled, this line will be signaled low 10ms before data is sent to the host over the UART.
- RF (RF Pad Version Only)
  The RF pin should be connected to a 50 Ohm Bluetooth ISM Band antenna.

UART Interface

The eb101 Universal Asynchronous Receiver Transmitter (UART) interface provides a simple mechanism for communicating with other serial devices using the R$232 protocol. While the standard R$232 protocol is used, the voltage levels are 0V to VDD_IO. Communications with a standard RS232 device would require an external R$232 transceiver IC.

When the eb101 is connected to another digital device, UARTRX and UART_TX transfer data between the two devices. The remaining two signals, UART_CTS and UART RTs, can be used to implement R$232 hardware flow control where both are active low indicators. All UART connections are implemented using CMOS technology and have signaling levels of 0V and VCC.

Power Supply

- GND

  All ground lines should be connected in parallel.

- VDD

  Positive supply for the module. Use of this line is optional if VDD_IO is supplied with 3.0V - 3.6V.

- VDD_IO
  Positive supply for 1/O and UART lines. If this line is supplied with 3.0V 3.6V then used of VDD is optional.
- NC
  For proper operation, all pins marked with NC should not be connected externally. All terminals should be placed on pads to ensure mechanical robustness

4.2.4 Bluetooth Software Stack

Overview

The eb101 module encapsulates the complexity of working with Bluetooth technology in order to make it simple to use and minimize the time required to add it to a product. The primary application profile that is supported is SPP, or the Serial Port Profile. This is the most popular and convenient protocol for many embedded applications of Bluetooth since it emulates a simple serial port link between devices. Once the connection is established, communications between the endpoints is the same as for a wired serial port. The ebl01 supports two main operating modes: Easy Connect and Commanded. When Easy Connect is used the device operates as a simple cable replacement solution. In Command Mode there is a rich set of functions that allow the host to have programmatic control over the module. In both modes the factory default communication parameters are 9600 Baud, 8 Data Bits, 1 Stop Bit, No Parity, and No Flow Control.

Easy Connect

Easy Connect mode provides a simple cable replacement solution that can be used without sending any commands to the device. A common implementation of this feature is to connect a momentary switch and LED to the eb101 for initiating pairing and monitoring the status of the module. Once the onetime pairing procedure is complete, data is transmitted between the devices automatically without the need for additional configuration or control.

The wireless cable connection will be established and maintained whenever the eb101 1S powered. This usage is most common when you want to enable a device with wireless technology, but do not want to make any modifications to it other than connecting it to an eb Serial device. Pairing two Easy Connect devices that use the common implementation is suite simple. Put each device into pairing mode by holding the Easy Connect button while u applying power to the units and then releasing it when the LED turns on.

The devices will locate and pair with each other automatically forming a reliable and secure wireless connection. When this process is complete and the devices are ready for use. the indicator LED will begin to blink slowly. The paired devices will automatically establish and maintain a secure wireless connection whenever they are powered on.

Command Mode

Command mode provides the host with programmatic control over the module and its configuration. There are a number of commands that can be sent to change the baud rate. locate other devices that are in range, check the firmware version, etc. All commands are sent using visible ASCII characters (123 is 3 bytes "123"). Upon the successful transmission of a command, the ACK string will be returned. If there is a problem in the syntax of the transmission then a NAK string is returned. After either the ACK or NAK, a carriage-return character is returned. When a prompt (followed by a .>0) is returned, it means that the eb101 radio is in the idle state and is waiting for another command.

White space is used to separate arguments of the command and a carriage-return (ASCII 0x0D) is used to mark the end of the command. Once the eb101 radio is connected to another Bluetooth device, all data written to the UART will be transmitted wirelessly to the remote device. Therefore, NO further commands can be issued until the eb101 radio is disconnected or switched back to command mode by use of the BREAK control line or the soft break command. The connection status line of the eb 101 module can be monitored to determine if there is currently an active connection. Command Set: The Embedded Blue command set is comprised of visible ASCII characters. Therefore, a command can be issued from a terminal application, such as HyperTerminal, or directly from a custom application program, written in a programming language such as assembly, C, Java, or Basic. From a microcontroller application, these commands can be Issued directly to the asynchronous UART on the device.

Command Set:

The Embedded Blue command set is comprised of visible ASCII characters. Therefore, a command can be issued from a terminal application, such as HyperTerminal, or directly from a custom application program, written in a programming language such as assembly, C, Java, or Basic. From a microcontroller application, these commands can be Issued directly to the asynchronous UART on the device.

4.2.5 Sample Command Scenario

The following diagram illustrates a possible command scenario. The data on the left is the data as it might appear in a terminal application that is directly connected to the A7 eb101 Serial Firmware based device such as the eb301 adapter. The data that is not shaded is the data that is being typed into the terminal to send to the eb101 and the shaded data is the data being received by the terminal from the eb101. The data on the right is a description of what is happening. The row breaks are given for clarity of description only.

>lst visible name 15<CR> ACK<CR>

00:0C:84:00:05:29 MyCellPhone <CR>

00:80:C8:35:2C:B8 JoesLaptop<CR>

00:0C:84:00:83:F9 eb101> > (List devices that are currently visible.)

con 00:0C:84:00:83:PF9<CR> ACK <CR>>

(Connect to a device. This data will be sent over the wireless link. Transmiting and receiving data over the Bluetooth connection. (The terminal example here shows that the link is full duplex).

<1 sec>+++<1sec>CR> >

(Temporarily go to command mode by waiting for at least I second, sending "+++", and delaying again for 1 sec.)

get addr remote<CR> ACK<CR> 00:0C:84:00:83:F9<CR>

(Get the address of the remote device.)

Ret <CR> ACK<CR>>

(Return to data mode. This text is being sent over the This text is being received wireless connection over the wireless connection Transmitting and receiving data over the Bluetooth connection.)

<1sec>+++<1sec><CR>>

Break into command mode).

dis <CR> ACK<CR>

(Disconnect the current data connection.)

4.2.6 Security

Bluetooth defines being able to see a device and being able to connect to a device as part of the security model. These features are exposed by the eb101 Serial Firmware through the 'set visible' and 'set connectable' commands. This is a very course level of control, but it is also quite effective and can be used in combination with other security features.

The eb101 Serial Firmware uses the "set security' command to configure access control. When security is turned off, connection attempts will be allowed from all remote devices. Forming a trusted relationship is carried out automatically in this mode the first time that a device attempts to establish a connection with the proper passkey. When security is turned on, only connections from trusted devices will be allowed and no new devices may become trusted.

CHAPTER-V

DESIGN AND DEVELOPMENT

5.1 UART Design

- The set of registers that control transmitters operation are
- TDRE--transmit data register empty
- TDR-Transmit data register
- TSR Transmit shift register
- The system waits until TDRE = '1' and then loads a byte of data into TDR and clears TDRE.
- The UART transfers data from TDR to TSR and sets TDRE
- The UART outputs a start bit("0") for one bit time and then shifts TSR right to transmit the eight data bits followed by stop bit('1').

5.1.1 Transmitte

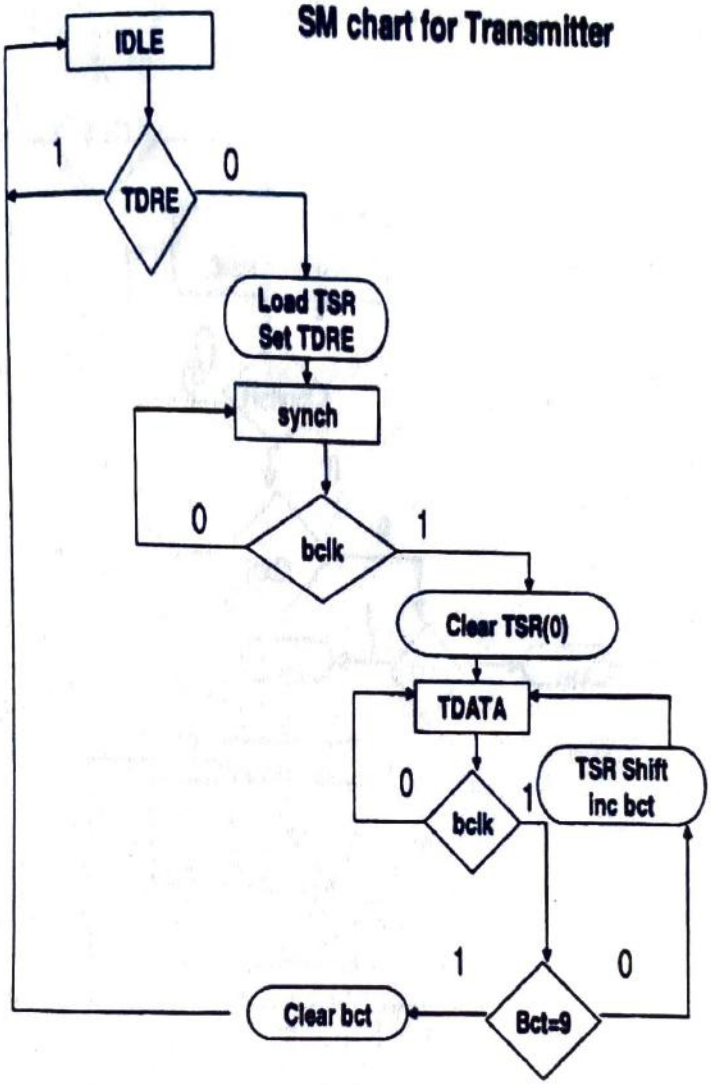


Fig5.1: SM chart for Transmitter

5.1.2 Receiver

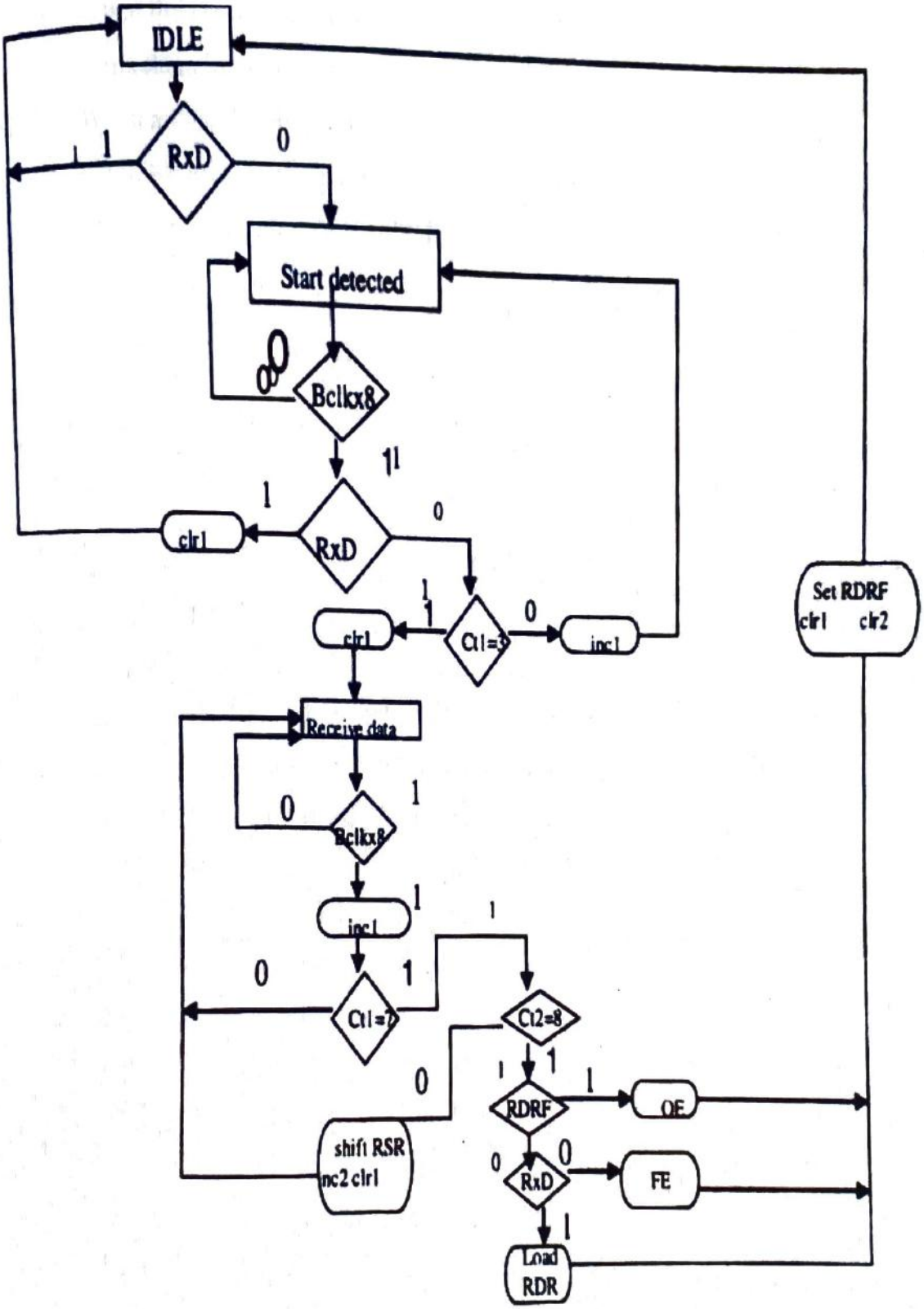


Fig5.2: SM chart for Receiver

The operation of the UART receiver is as follows:

1. When the UART detects a start bit, it reads in the remaining bits serially and shifts them into the RSR.
2. When all the data bits and the stop bit have been received, the RSR is loaded into the RDR, and the Receiver Data Register Full(RDRF) lag in the SCSR is set.
3. The microcontroller checks the RDRF flag, and if it is set, the RDR is read and the flag is cleared.

5.2 Interfacing the FPGA to Bluetooth

C code algorithm:

Algorithm for the program:

1. start
2. 2. send cmd * list visible devices"
3. if ACK is received, goto 3. else 1
4. receive addresses of devices from the port
5. display the received add for user convenience
6. let the user choose one of the displayed addresses
7. send cmd "CON (user inputted add)" to the port
8. if ACK received goto 8, else goto 6
9. set the comp in transmission mode
10. transmit data
11. if ACK is received goto 11, else goto 9
12. send "disconnect (user inputted add)" to the port
13. if ACK is received goto 13, else goto 11
14. end

CHAPTER-VI

Experimental Results

6.1 Configuring FPGA for UART waveforms

6.1.1 Transmitter

Input waveform

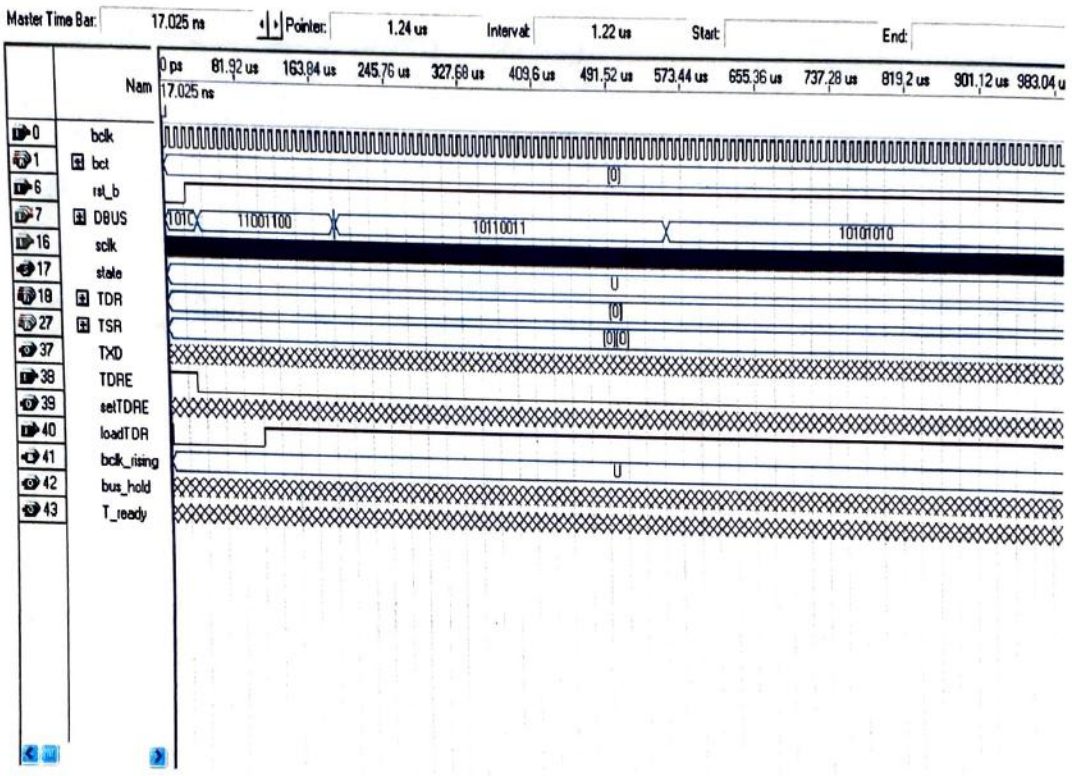


Fig 6.1 Transmitter input

Output waveforms

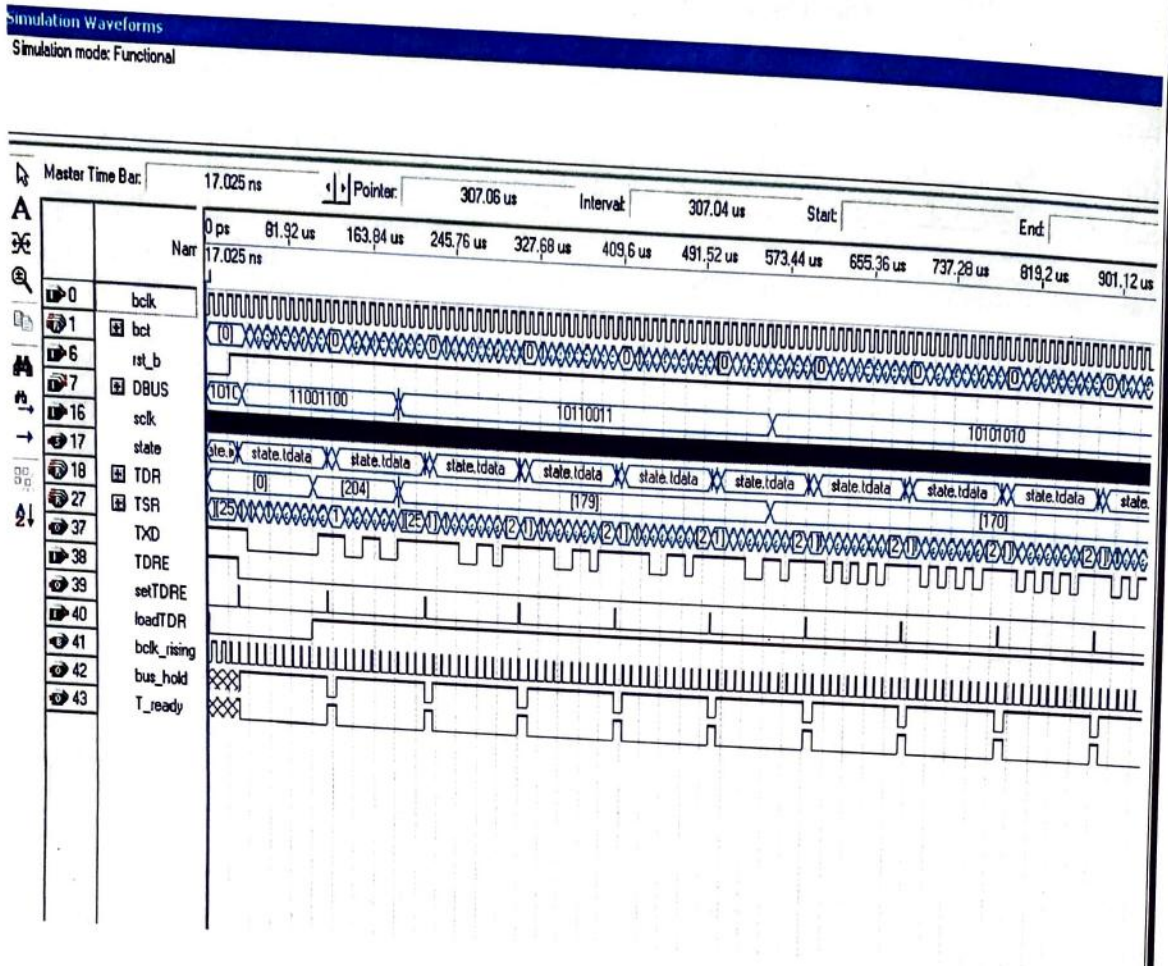


Fig 6.2 Transmitter output

Detail report

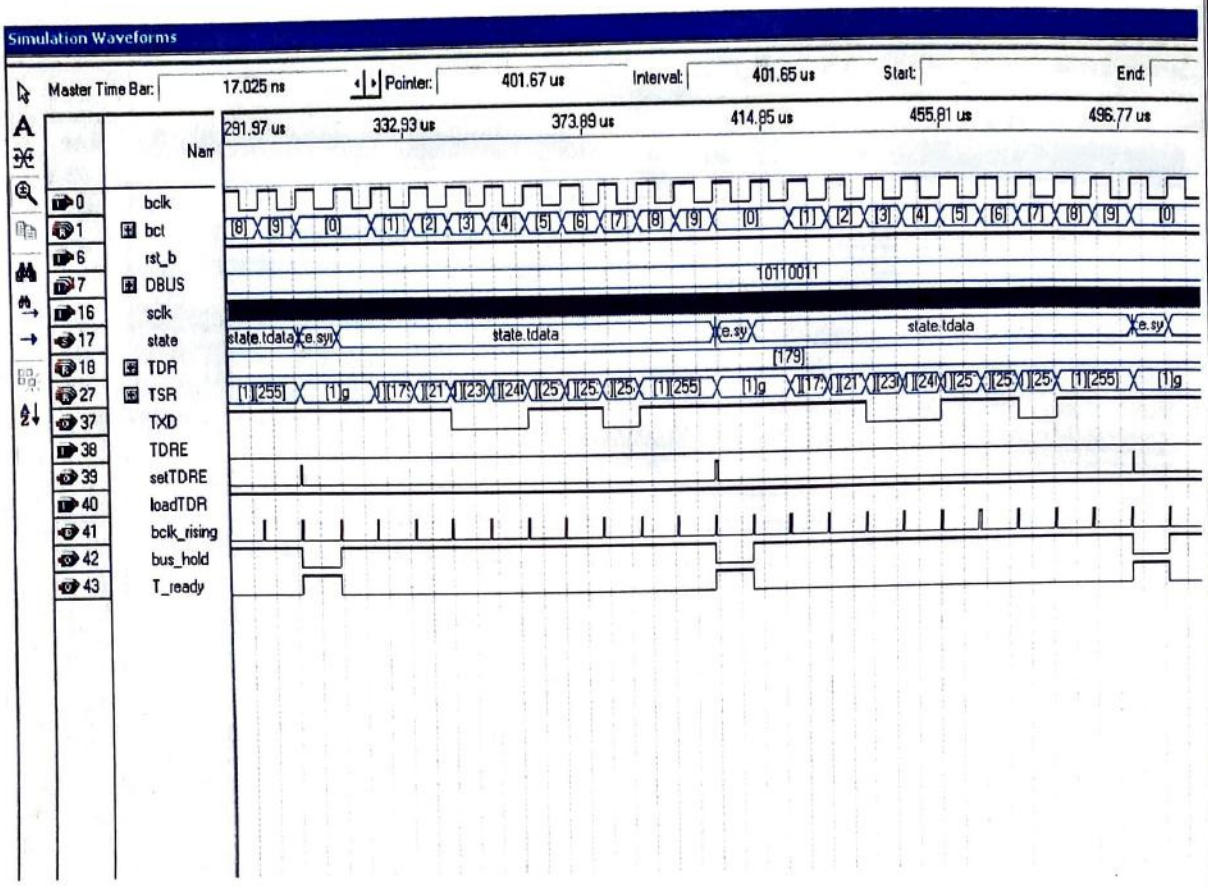


Fig 6.3 Detailed Report of Transmitter

6.1.2 Receiver

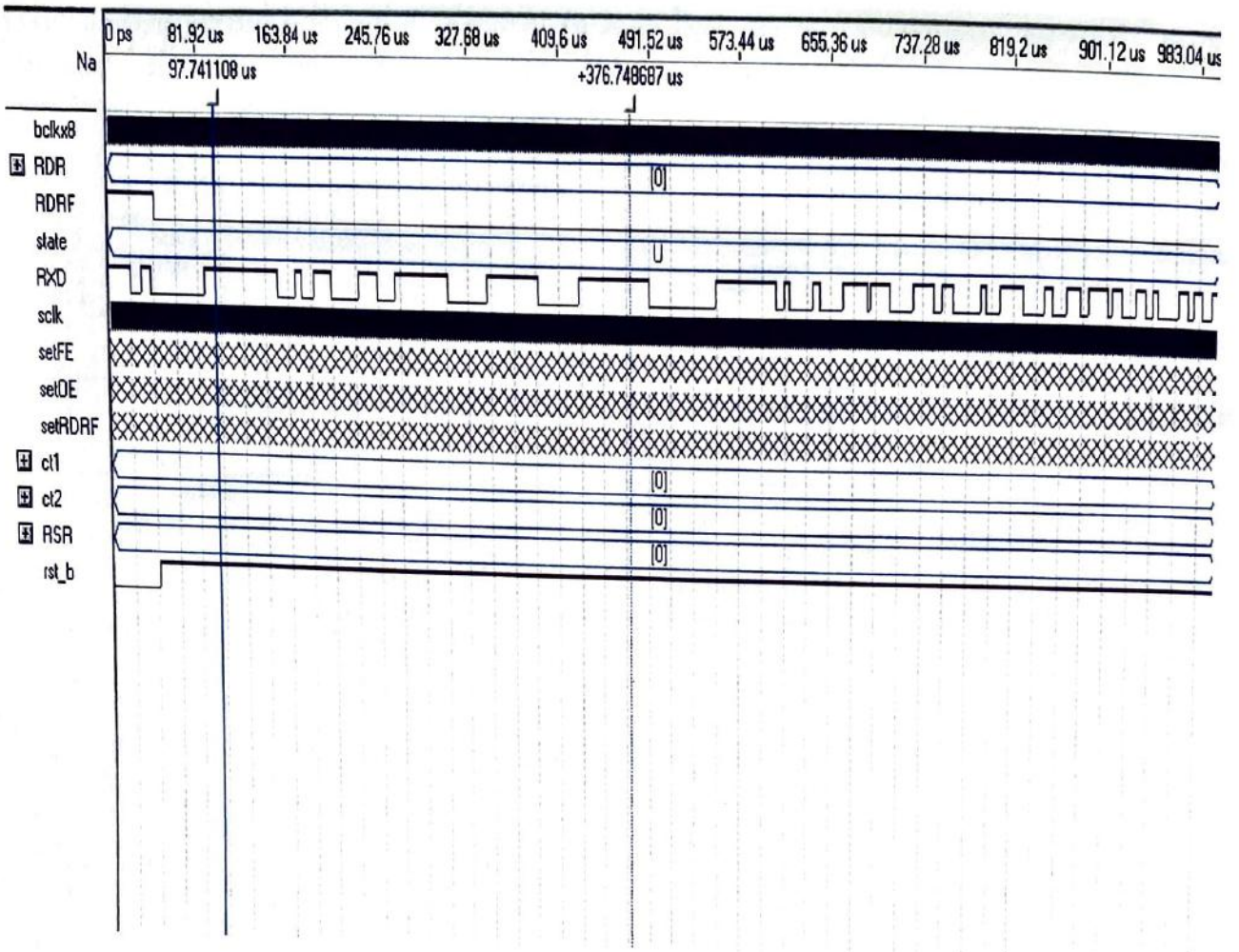


Fig 6.4 Receiver input

Simulation report

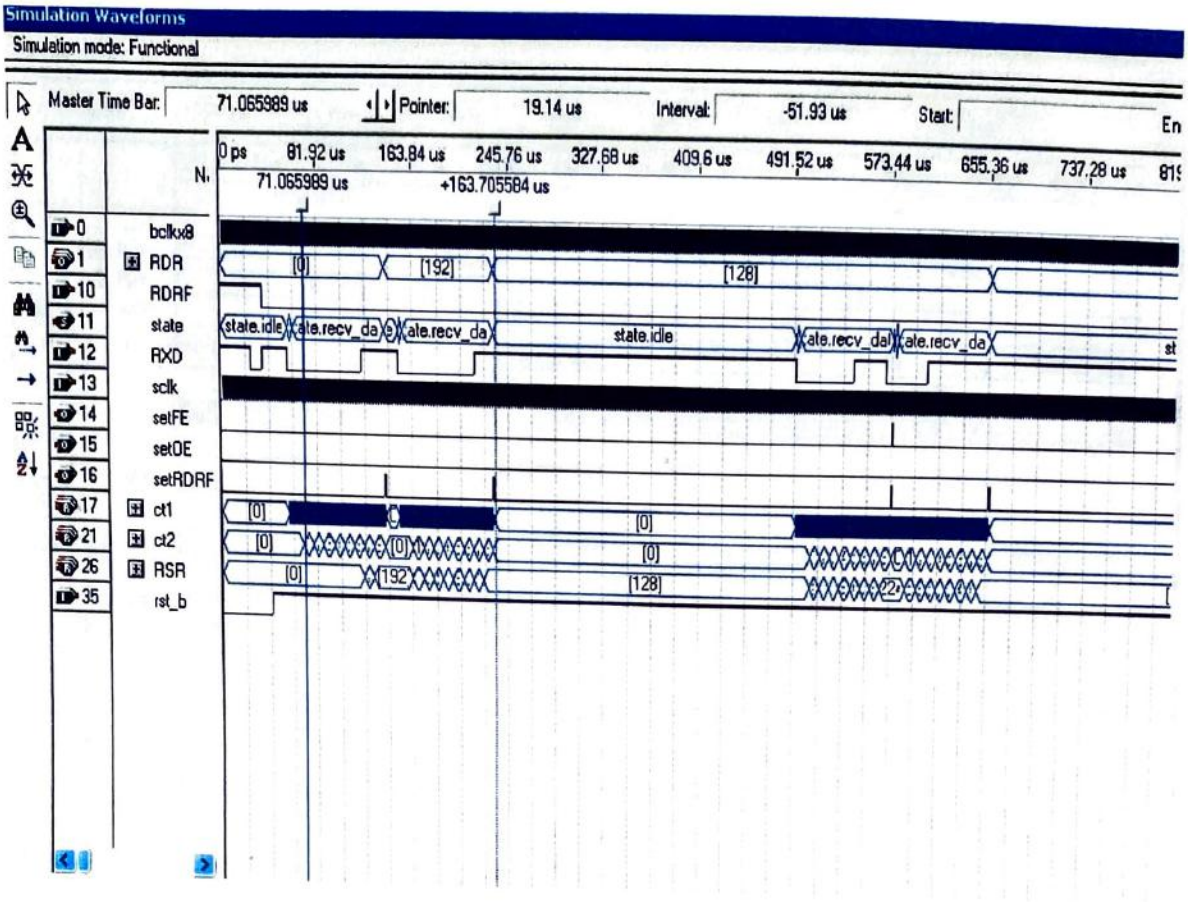


Fig 6.5 Receiver output

Detailed report

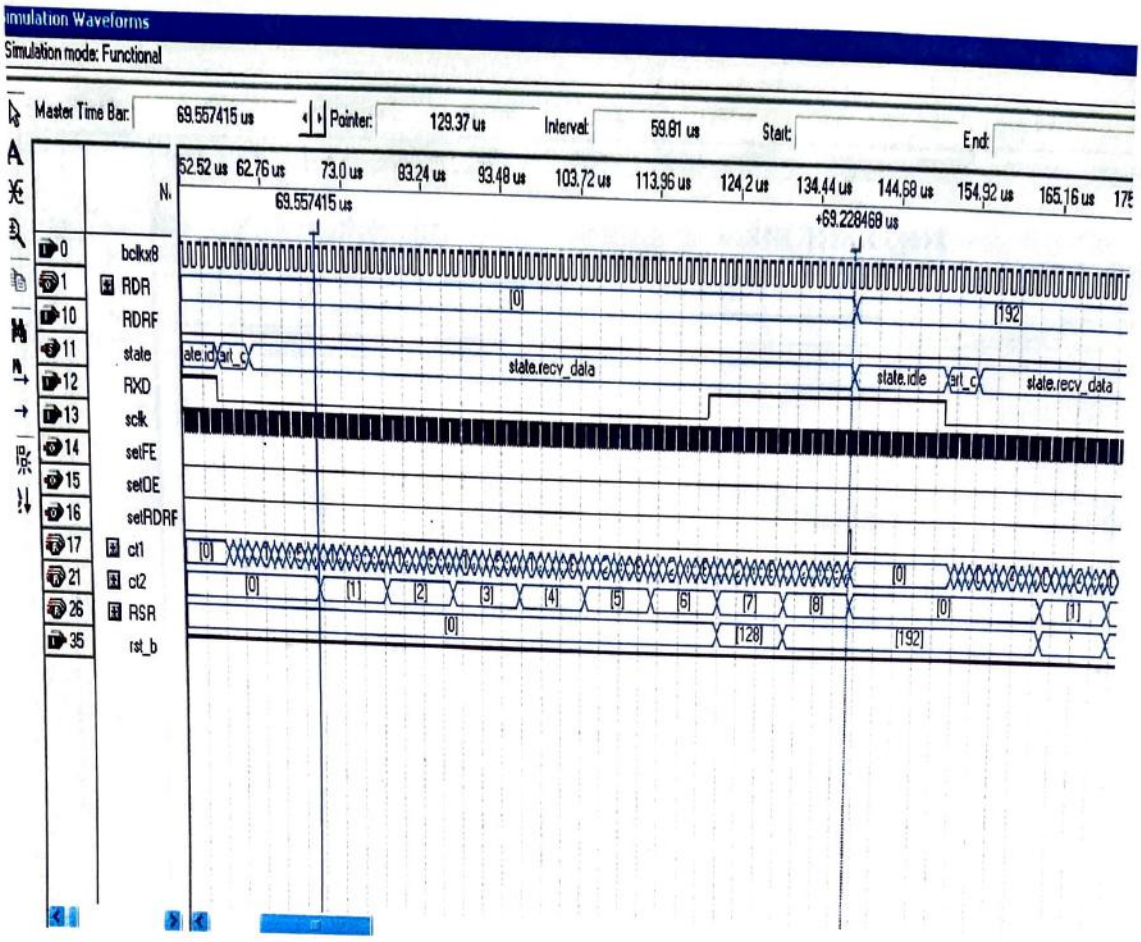


Fig 6.6 Detailed Report of receiver

6.1.3 Baud rate generator

Input waveform

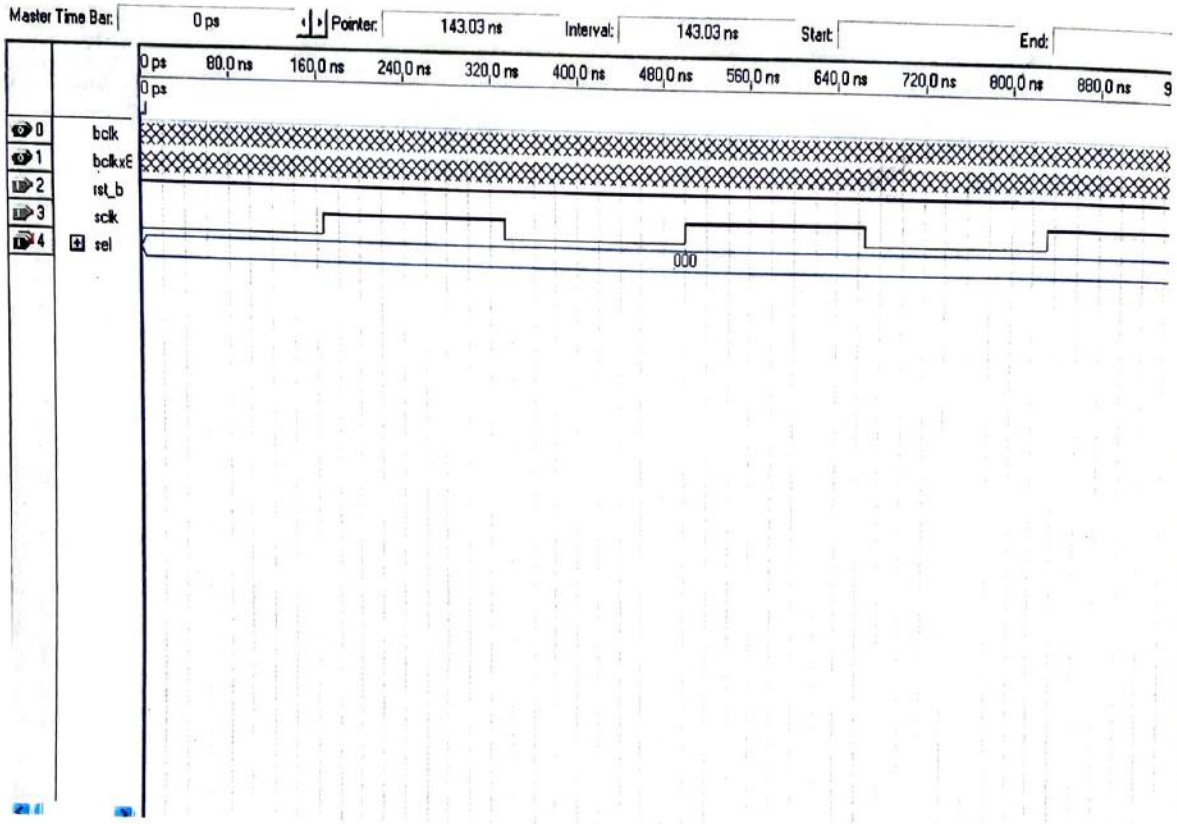


Fig 6.7 Baud Rate Generator input

Output waveform

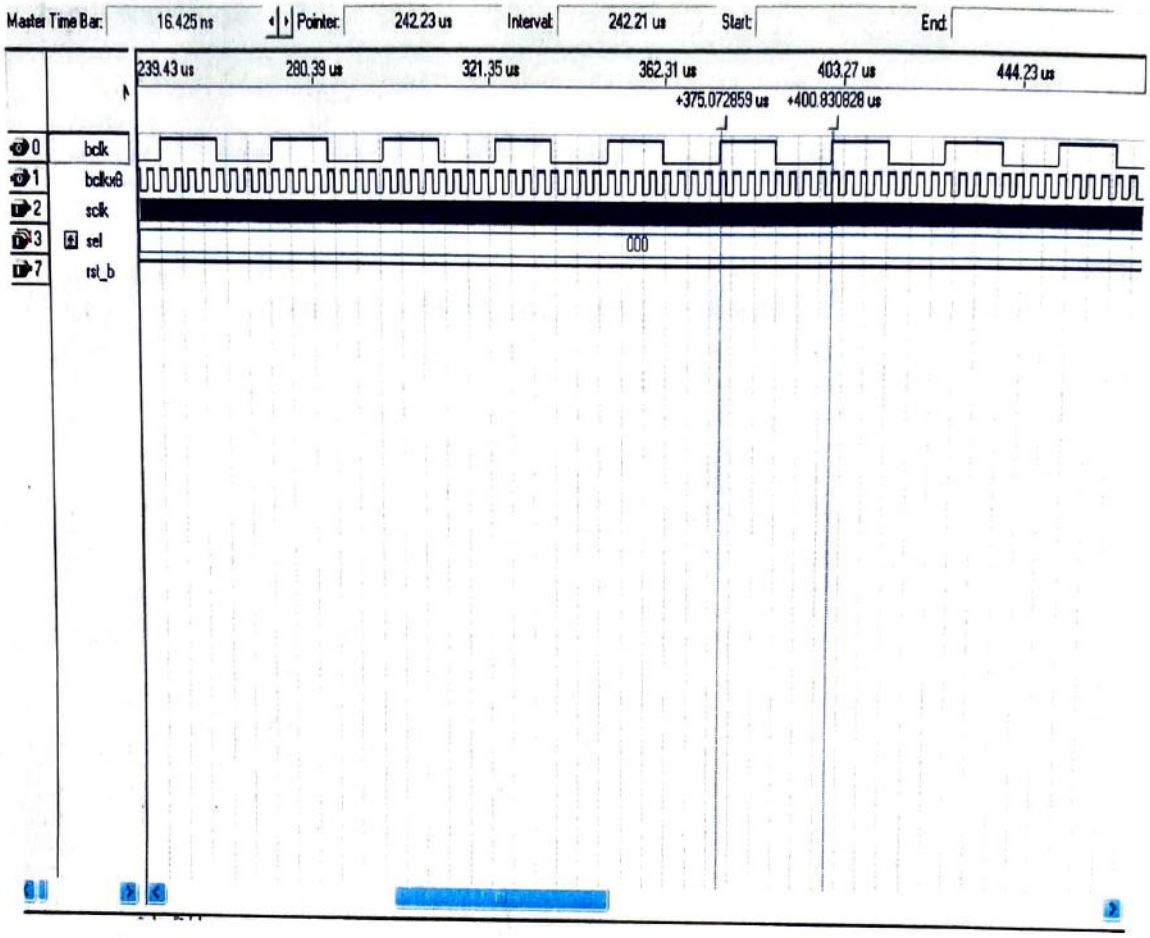


Fig 6.8 Baud Rate Generator output

6.1.4 UART

Input waveform

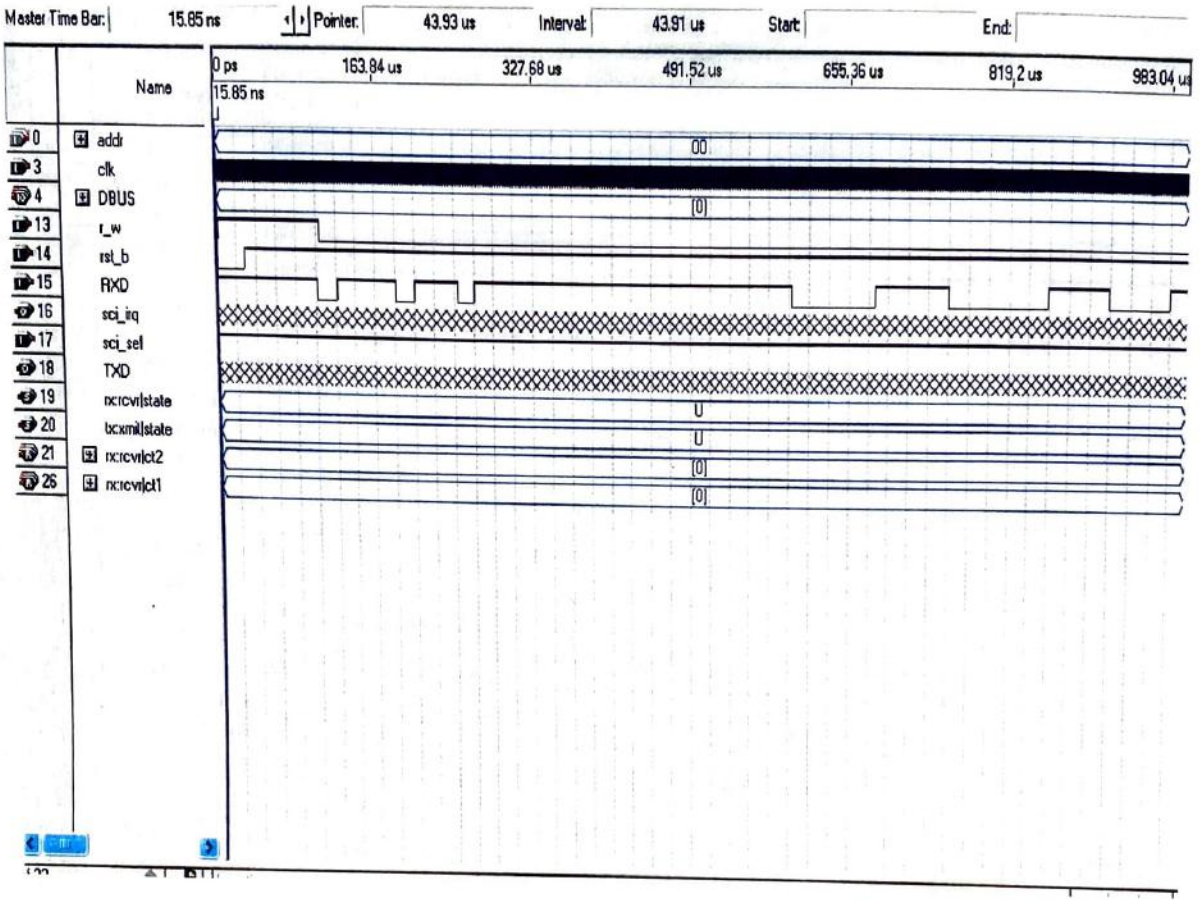


Fig 6.9 UART input

Output waveform

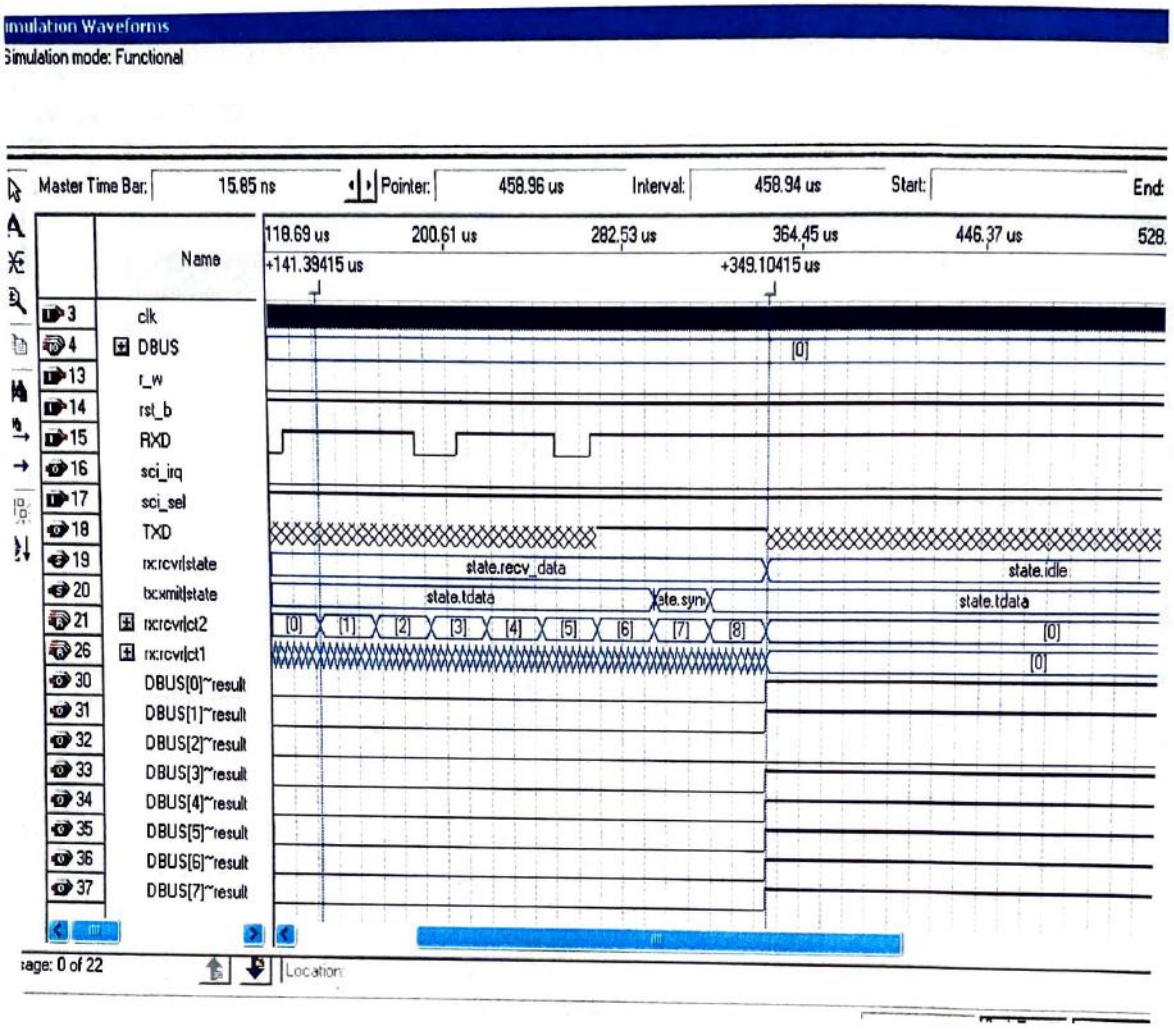


Fig 6.10 UART output

6.2 Bluetooth Transmission

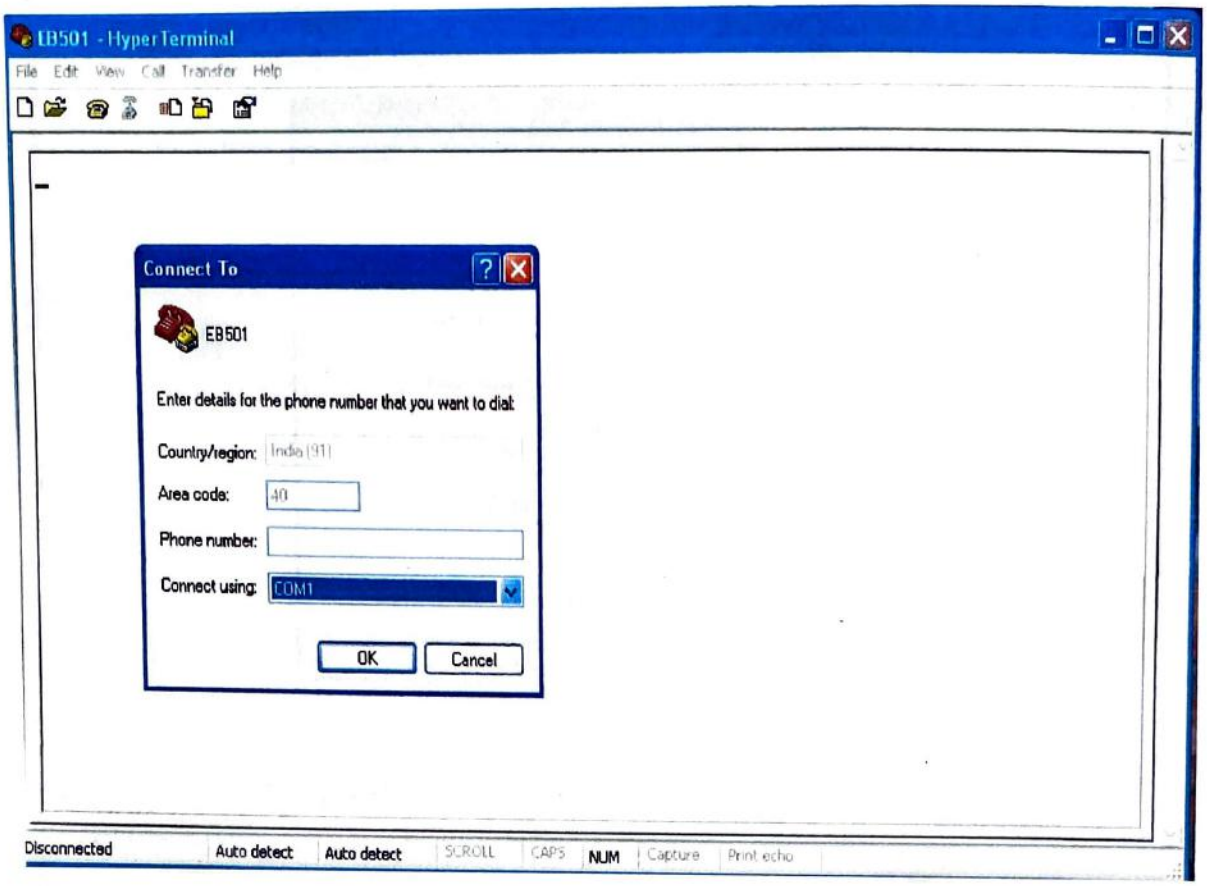


Fig 6.11 Hyper Terminal connection

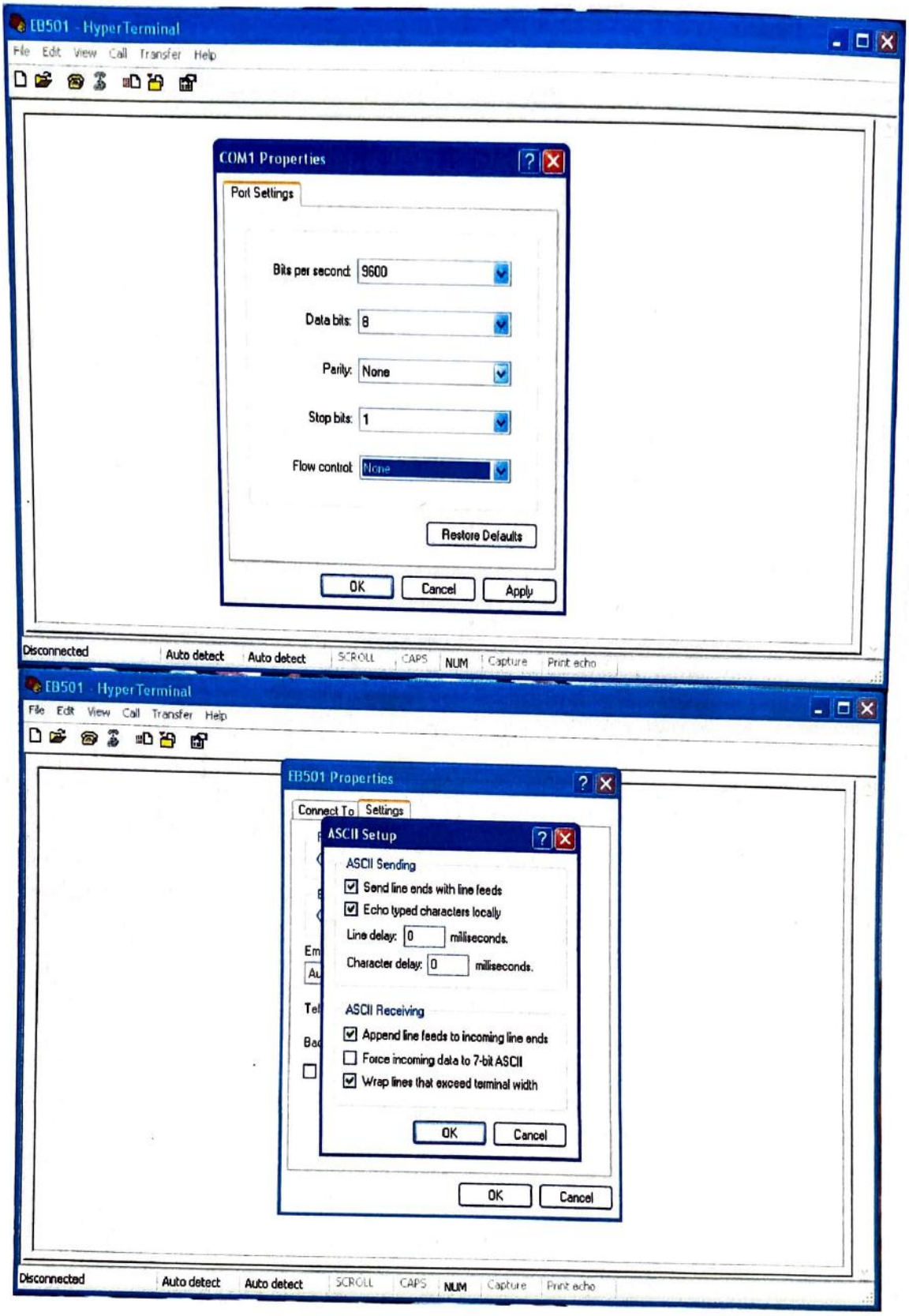


Fig 6.12 Enabling comport and ASCII setup

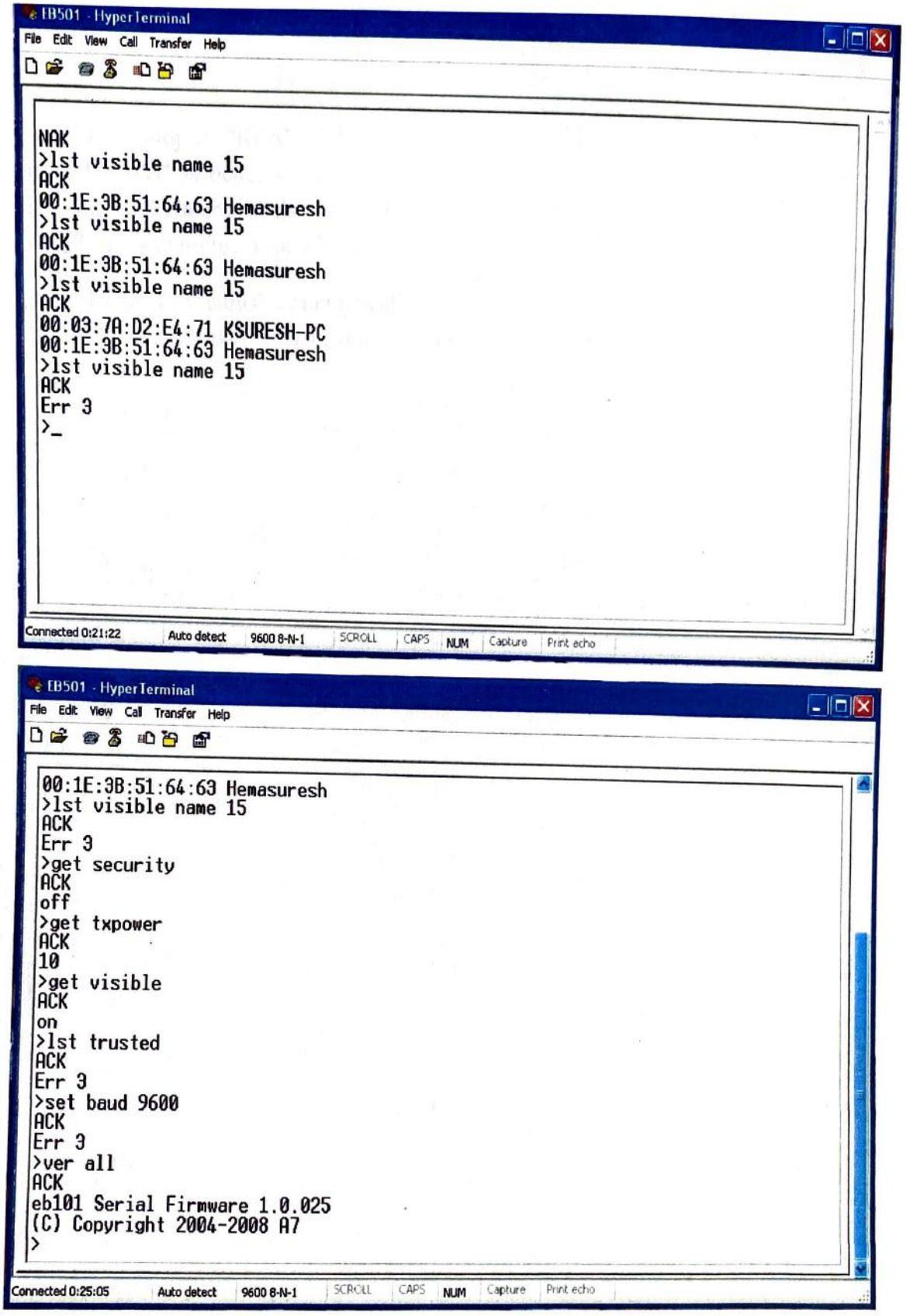


Fig 6.13 Hyper Terminal output

CONCLUSION

This project "REAL TIME TRANSFER OF DATA FROM RLG USING BLUETOOTH" is better version of existing wired system ,replasing wires with more fiexibility and features. This system transfers the live data from Ring Laser Giyro to PC using Bluetooth protocol for wireless transmission.

Hence to establish communication between RLG and PC ,the fpga is configured for UART and comport is enabled for serial communication.

REFERANCES

- Digital System Design through VHDL by CHARLES.HROTHJr
- FPGA prototyping by VHDL examples by PONG P.CHU
- http://www.bluetooth.com/bluetooth/products
- http://www.bevondlogic /seial/serial.htm
- http://www.beyondlogic /serial/usb
- http://www.beyondlogic /serial/usb/41
- http://en.wikipedia.org/wiki/ring.laser. gyroscope
- http://en.wikipedia.org/wiki/surface-mounttechnolog
- http://en.wikipedia.org/wiki/serial.port
- http:// www.a7eng.com

Appendix-A

CODING

VHDL Coding for configuring the FPGA for UART

Transmitter

```
library ieee:
use ieee.std_logic_I 164.all;
entity tx is
        port(bclk.sclk,rst_b,TDRE,loadTDR:in std_logic;
                DBUS:in std_logic_vector(7 downto 0);
                setTDRE,TXD:out std_logic:
                bus_hold,T_ready:out std_logic);
end tx,
architecture xmit of tx is
        type statetype is(IDLE,SYNCH,TDATA):
        signal state,nextstate:statetype;
        signal TSR:std_logic_vector(8 downto 0); -----transmit shift register
        signal TDR:std_logic_vector(7 downto 0)---transmit data register
        signal bct:integer range to 9, counts number of bits sent
        signal inc.clr,loadTSR,shftTSR,start,TSRout:std_logic;
        signal bclk_rising.bclk_dlayed:std_logic;
begin
TXD<=TSR(0);
setTDRE<=loadTSR;
bclk_rising<=bclk and (not bclk_dlayed);
-indicates the rising edge of bit clock Xmit_control:process(state,TDRE,bct,bclk_rising)
```

```
Begin
inc='0'; clre="0;load'TSR<='0';shftTSR<="0':start<="0';        resetcontrol signals
case state is
 when IDLE=>
        if (TDRE=0')then
        loadTSR<=l; nextstate<=SYNCH: bus_hold<=0';T_ready<="1;
        else nextstate<=IDLE:
 end if;
 when SYNCH=>
        if(bclk_rising="'l")then
         start="'l'; nextstate<=tdata;
        bus_hold=0;T_ready<="1':
        else nextstate<=SYNCH;
 end if;
 when TDATA=>
        if(bclk_rising=0') then nextstate<=TDATA;
        elsif(bct/=9)then shftTSR<=1;inc<='1';nextstate<=TDATA;
        bus_hold=1':T_ready<=0;
        else cir <= 'l'; nextstate <= IDLE;
        end if;
 end case;
end process;
xmit_update:process(sclk.rst_b)
 begin
        if(rst_b=0")then
                TSR<="1|1111111":state<=IDLE;bct<=0;bclk_dlayed<='0';
```

```
        elsif(sclk'event and sclk='1")then
                state<=nextstate;
        if(clr="1') then bct <=0;elsif(inc="1 )then
                bct<= bct+1;
        end if;

        if(loadTDR="1) then TDR<=DBUS;
        end if;
        if(loadTSR=1) then TSR<<=TDR & '1;
        end if;
        if(start="1) then TSRout <= '0;
        end if;
        if(shfTSR=1) then TSR<=l'&TSR(8 downto 1);
        end if;
-----shift out 1 bit
        bclk_dlayed<=bclk;--bclk delayed by I sysclk
        end if;
   end process ;
end xmit;
```

Receiver

```
library ieee;
use ieee.std_logic_1 164.all;
entity rx is
        port(RXD.bclkx8,rst_b,sclk,RDRF:in std_logic:
         RDR:out std_logic_vector(7 downto 0);
        setRDRF.setOE. setFE: out std_logic);
end rx;
architecture revr of rx is
type statetype is(idle,start_detected,recv_data);
signal state,nextstate: statetype;

        signal RSR :std_logic_vector(7 downto 0);--recieve shift register
        signal ctl :integer range 0 to 7;-indicates when to read the rxd
        signal ct2 :integer range 0 to 8--counts number of bits read
        signal incl ,inc2.clrl,clr2,shftRSR, loadRDR:std_logic;
        signal bclkx8_dlayed, bclkx8_rising:std_logic;
begin
bCIkx8_rising<= bclkx8 and(not bclkx8_dlayed),--indicates rising edge of bitx8
revr_control:process(state,RXD,RDRF.ctl.ct2.bclkx8_rising)
        begin
        ---- reset control signals inci<=0:inc2<=0'; clrl<=0'; clr2<=0;
        shftRSR<="0: loadRDR<=0:setRDRF«=0':setOE<=0:setFEk=0;
case state is
when idle=>
        if(RXD=0) then nextstate<=start_detected;
```

```
        else nextstate <= idle;
        end if;
when start_detected =>
        if(bclkz8_rising= 0') then nextstate <= start_detected;
        elsif(RXD=1) then clrl<='l; nextstate<=idle;
        elsif(ctl=3) then clrl="T; nextstate<=recv_data;
        else incl<="l; nextstate<=start_detected;
        end if;
 when recv_data =>
        if (bclkx8_rising=0') then nextstate<= recv_data;
        else incl<='1';
        if(ctl=7)then nextstate<=recv_data;
        ----Wait for eight clock pulses
        elsif(ct2/=8)then
        shftRSR«=1; inc2<="1; clrl<="l'-----read nxt data bit
        nextstate<=recv_data;
        else
        nextstate <= idle;
        setRDRF<="1; clrl<="'1; clr2«='1';
        if(RDRF="1) then setOE<=l:-overrun error
        elsif(RXD=0") then setFE<='|; -framing error
        else loadRDR <=°1';
        end if;
        end if;
 end if;
```

```
        else nextstate <= idle;
        end if;
when start_detected =>
        if(bclkz8_rising= 0') then nextstate <= start_detected;
        elsif(RXD=1) then clrl<='l; nextstate<=idle;
        elsif(ctl=3) then clrl="T; nextstate<=recv_data;
        else incl<="l; nextstate<=start_detected;
        end if;
 when recv_data =>
        if (bclkx8_rising=0') then nextstate<= recv_data;
        else incl<='1';
        if(ctl=7)then nextstate<=recv_data;
        ----Wait for eight clock pulses
        elsif(ct2/=8)then
        shftRSR«=1; inc2<="1; clrl<="l'-----read nxt data bit
        nextstate<=recv_data;
        else
        nextstate <= idle;
        setRDRF<="1; clrl<="'1; clr2«='1';
        if(RDRF="1) then setOE<=l:-overrun error
        elsif(RXD=0") then setFE<='|; -framing error
        else loadRDR <=°1';
        end if;
        end if;
 end if;
```

```
end case;
end process;

revr_update:process(sclk,.rst_b)
begin
        if(rst_b=0) then state<=idle;bclkx8_dlayed<=0';
                ctl<=0;ct2<=0;
        elsif(sclk'event and sclk='1')then
                state<=nextstate;
        if(clr 1='1')then ct 1<=0;
                elsif(inc 1='1')then
                ct 1<=ct 1+1;
        end if;
        if(clr 2='1')then ct 2<=0;
        elsif(inc 2='1')then
        ct 2<=ct 2+1; end if;
        if(shftRSR='1') then
                RSR<=RXD & RSR(7 downto 1)--update shift register
        end if;
        ifloadRDR=1) then RDR<=RSR;
        end if;
        bclkx8_dlayed<=bclkx8-bclk;-------- delayed by 1 clk
end if;
end process;
end rcvr;
```

Baud Rate generator

```
library ieee;
use ieee.std_logic_I 164.all;
use iee.std_logic_unsigned.all;- use + operator, CONVERT_INT func use
iee.std_logic_arith.CONV_STD_LOGIC_VECTOR;
entity grb is
        port (sclk.rst_b:in std_logic;
                sel :in stdlogic_vector(2 downto 0);
                bclkx8:buffer std_logie;
                bclk:out std_logic):
        end grb;
architecture baudgen of grb is
signal tl: std_logic_vector(3 downto 0):="0000"; --divided by 13 counter

Signal t2: std_logic_vector(7 downto 0):="00000000":--divide by 256 counter

signal t3: std_logic_vector(2 downto 0):="000";--divide by 8 counter

signal clkdiv17: std_logic;
begin
process(sclk) --first divide system clk by 17
begin
        if(sclk'event and sclk="1) then
        if(t="1100") then tl <= "0000";
        else tl <=tl +1;
end if;
```

```
end if;
 end process;
clkdivi7<= ti(3);
 process (cikdiv17)
 begin
        if(rising_edge(clkdiv17)) then
        t2<=t2+1;
         end if; end process;
 bclkx8= t2(CONV_INTEGER(sel);        --sel baud rate
 process (bclkx8)
 begin
         if(rising_edge(bclkx8)) then t3<=t3+1;
         end if;
 end process;
 bclk =t3(2);   -- Bclk is Bclkx8 divided by 8
 end baudgen;
 4.1.4 UART; library ieee;
 use ieee.std_logic_1164.all;
 use ieee.std_logic_unsigned.al;     -- use +' operator, CONVERT_INT func use
ieee.std_logic_arith.CONV_STD_LOGIC_VECTOR;
 entity uart is
        port(sci_sel.r_w.clk.rst_b,RXD:in std_logic;
              addr:in std_logic_vector(! downto 0);
              DBUS :inout std_logic_vector( 7 downto 0);
        sci_irq.TXD :out std_logic);
 end uart;
```

```
architecture uartl of uart is


component rx
        port(RXD,bclkx8,rst_b,sclk, RDRF:in std_logic;
                RDR:out std_logic_vector(7 downto 0);
                setRDRF.setOE,setFE:out std_logic);
end component;


component tx
        port(bclk,sclk.rst_b,TDRE, loadTDR:in std_logic;
                DBUS:in std_logic_vector(7 downto 0);
        setTDRE,TXD:out std_logic);
end component;


component grb
        port (sclk.rst_b:in std_logic;
        sel :in std_logic_vector(2 downto 0);
        bclkx8:buffer std_logic;
        bclk:out std_logic);
end component;
signal RDR: std_logic_vector(7 downto 0)recieve data register
signal SCSR: std_logic_vector(7 downto 0);-status register
signal SCCR: std_logic_vector(7 downto 0)control register
Signal TDRE,RDRF,OE,FE,TIE,RIE:std_logic;
Signal baudsel:std_logic_vector(2 downto 0);
```

```
signal setTDRE,setRDRF,setOE,setFE, loadTDR,loadSCCR:std_logic:
signal clrRDRF,bclk,bclkx8,sci_read,sci_write:std_logic:

begin
rcvr : rx port map(RXD,bclkx8,clk,rst_b,RDRF,RDR,setRDRF,setOE,setFE):
x mit : tx port map(bclk,clk,rst_b,TDRE, load TDR,DBUS,setTDRE, TXD);
baudgen : grb port map(clk,rst_b,baudsel, bclkx8.bclk);
------this process updates the control and status registers
process(clk,rst_b)

 begin
        if(rst_b=0)then
                TDRE="1:RDRF<=0;OE<=0;FE<=0;
                TIE<=0;RIE<=0';
        elsif(rising_edge(clk))then
                TDRE=(setTDRE and not TDRE) or ( not loadTDR and TDRE);
                RDRFk=(setRDRF and not RDRF) or (not clrRDRF and RDRF);
                OE=(setOE and not OE) or (not cirRDRF and OE;
                FE<=(setFE and not FE) or (not clrRDRF and FE);
        if(loadSCCR='1) then TIE«=DBUS(7);RIE«=DBUS(6);
                baudselk=DBUS(2 downto 0);
         end if;
  end if;
 end process;
```

```
--- irq generation logic
sci_irq<="|' when ((RIE=1l' and (RDRF="1' or OE=1))
        or(TIE="1' and TDRE='1))
        else '0;
--bus interface

SCSR<=TDRE & RDRF & "0000" & OE & FE;
SCCR<=TIE & RIE & "CO0" & baudsel;
sci_read<'l when (sCi_sei="|and r_w=0') else '0';
sci_write<='l' when (sci_sel="|' and r_w='1) else 0';
clrRDRF='1' when (sci_read ='|' and addr="00")else '0';
loadTDRe='l' when ( sci_write="1' and addr="00")else 0';
loadSCCR<="| when ( sci_write="l' and addr="10")else '0';
DBUS="ZZZZZZZZ" when (sci_read ='0')--------ristate bus when not reading
else RDR when (addr="00")------WRITE APPROPRIATE REGISTER TO BUS
else SCSRR when (addr="01")
 else SCCR;
end uart;
```

C Code

```
#include<stdio.h>
 #include<bios.h>
#inciude<conio.h>
#include<dos.h>
#define COMI 0
#define DATA_READY Ox 100
#define SETTINGS (0xEO | Ox03 |0x00 | 0x00) int main (void)
{
int in.y; int i=0; int J;
int status; int length;
static char BTCOMMAND[] = "Ist"; char out;
char out 1[30]; char out2[30];
length =strlen(BTCOMMAND); clrscr();
bioscom(0,SETTINGS.COM1); /*Initialize the port*/ sleep(2);
for(j=0;j<length;j++)
{
Out 1[j]=BTCOMMAND[j];
bioscom(1,out 1[j],COM 1);/*Output a data*/ gotoxy(10,10);
textcolor(3);
printf("Data send     ");
 putch(out 1[j]); sound(4000);
//delay( 100); nosound();
}
bioscom(1. 13,COM1); /* Output Number 13*/ delay(100);
```

```
bioscom(0,SETTINGS,COMI); /*Initialize the port*/
 while(1)
{
        status = bioscom(3,0,cOM1); if (status& DATA_READY)
        {
        if((out= bioscom(2,0,COM1) & 0x7F) !=0) /* INput a Data*/
                {
                out2[i++] = out;
                 gotoxy(20,20);
                textcolor(3);
                printf("Data Received-----  ");
                delay(600);
                putch(out);
                sound(4000);
                delay(100);
                nosound();
                if (out == 13)
                {
                        printf("\nin CR Detectedin End of command\n\nPress any key to
                continue/");
                         break;
                }
         }
  }
```

```
        if (kbhit())
        {
        if (in = getch()) == 27)
        break;
        }
getch();
return 0;
```