2. State of art technology

The project aims at developing a real time system for establishing the dumb communication. For the success of the project it is equally important to study about the existing techniques which are helping the dumb to communicate. The state of art technology refers to the existing methodologies for achieving this aim. Owing to the ever increasing needs of the society, day by day technology is improving. Many of the present day researches are undertaken with the aim of building some socially relevant products which may help the people to advance technologically.

Even after all these developments in the technology, it's a bitter fact that there has not been any significant works done in helping the dumb people to communicate. Even today we depend on a translator to understand what the dumb people are trying to convey. Translators are people who can understand the sign language of the dumb. They manually decode their signs and convey the message to the world. But unfortunately in most of the cases such a translator will be absent and it makes the dumb another class of the society who have no methods to atleast convey their messages like other normal human beings.

Thus we can say that the state of art technology revolves around the presence of a human translator. This inspired us to think of a technology which can be handy, easy to use as well as efficient in its function. In this project we are trying to build such a product.

2.1 Recent developments

Owing to the social relevance of this topic fortunately in the recent years some developments have been witnessed. The most recent research on this topic uses the visual recognition of static gesture in which the recognized hand gestures are obtained from the visual images on a 2 D image plane. The processing occurs in MATLAB wherein different algorithms are used for the processing of the image. The main disadvantage of the system is its size and time consumption. Compatibility with different versions of MATLAB also became a problem.

Thus the recent development in this field includes the usage of a image acquisition system and the processing unit for processing the gestures and recognizing them correctly. The development in the field of artificial neural network has also played a very significant role in the

development of ANN models which can identify the gestures correctly and produce an output. Even Hidden Markov Models are used for the purpose.

The system that we are proposing here is different from the existing techniques since it aims in developing a completely independent, self sustained system which in future can be developed onto a complete product. No additional processors or external units are essential here. The system is independent by all means. The long time aim of the project is to develop an integrated chip which can help the dumb to exist in the society like a normal human being.

3. Literature Review

The success of any project lies in the understanding about the needs and techniques of the project. Hence we started with the literature review of the techniques available to implement the project. We have selected the FPGA module to be Cyclone II provide by Altera. VHDL (Very high speed Integrated Circuit Hardware Description Language) has been selected as the description language to model the hardware. All complex logic circuitries can be easily described using this language. As a part of the project so far we have reviewed the following papers and journals.

In [1] the problem of gesture recognition is narrowed down to that of hand gesture recognition and specifically deals with finger count extraction to facilitate further processing using the control so effected. A hand gesture recognition system has been developed, wherein the finger count in a certain input hand image is computed in accordance with a simple yet effective procedure. The problem of hand gesture recognition is solved by means of adopting a lucid albeit efficient algorithm which has been implemented using the Xilinx System Generator software tool. The algorithm followed is invariant to rotation and scale of the hand. The approach involves segmenting the hand based on skin color statistics and applying certain constraints to classify pixels as skin and non skin regions.

A different method had been developed for Hand Gesture Recognition for Indian Sign Language which consisted the use of Camshift and HSV model and then recognizing gesture through Genetic Algorithm. In that, applying camshift and HSV model was difficult because making it compatible with different MATLAB versions was not easy and genetic algorithm takes huge amount of time for its development. [2].

In [3] in order to get hand gesture feature vectors, the system adopts a vision-based hand tracking approach by using hand gesture segmentation algorithm. The system downloads those feature vectors data from large hand gesture feature vectors data base into the on-chip cache memory of an AP (Associative processor), then performs gestures matching in an extremely short time. Although gestures recognition processing is computationally very expensive by

software, latency free recognition becomes possible due to the highly parallel maximum-likelihood matching architecture of the AP chip.

In [4] Byung - woo min et al, presented the visual recognition of static gesture or dynamic gesture, in which recognized hand gestures obtained from the visual images on a 2D image plane, without any external devices. Gestures were spotted by a task specific state transition based on natural human articulation.

In [5] a method had been developed by P Subha Rajan and Dr G Balakrishnan for recognizing gestures for Indian Sign Language where the proposed that each gesture would be recognized through 7 bit orientation and generation process through RIGHT and LEFT scan. The following process required approximately six modules and was a tedious method of recognizing signs.

A method had been developed by T. Shanableh for recognizing isolated Arabic sign language gestures in a user independent mode. In this method the signers wore gloves to simplify the process of segmenting out the hands of the signer via color segmentation. The effectiveness of the proposed user-independent feature extraction scheme was assessed by two different classification techniques; namely, K-NN and polynomial networks. Many researchers utilized special devices to recognize the Sign Language. [6].

4. Principle

4.1 Hand gestures

Gestures are a form of nonverbal communication in which visible bodily actions are used to communicate important messages, either in place of speech or together and in parallel with spoken words. Gestures include movement of the hands, face, or other parts of the body. Physical non-verbal communication such as purely expressive displays, or displays of joint attention differ from gestures, which communicate specific messages. Gestures are culture-specific and can convey very different meanings in different social or cultural settings. Gesture is distinct from sign language.

To establish a communication or interaction with deaf and mute people is of utter importance nowadays. These people interact through hand gestures or signs. Gestures are basically the physical action form performed by a person to convey some meaningful information. Gestures are a powerful means of communication among humans. In fact gesturing is so deeply rooted in our communication that people often continue gesturing when speaking on the telephone. There are various signs which express complex meanings and recognizing them is a challenging task for people who have no understanding for that language.

4.2 The Indian sign language

The Sign language is very important for people who have hearing and speaking deficiency generally called deaf and mute. It is the only mode of communication for such people to convey their messages and it becomes very important for people to understand their language. Indian sign language is set of signs used in India. Different nations follow different signs as per their areas of interest and actions used. The government of India has even started an official website which gives us information about the signs used by the dumb in India and also includes lots of activities which helps the dumb to learn and live well in a society.

We have used the following gestures:

Fig. 4.1 Gesture for stop

Fig. 4.2 Gesture for okay

Fig. 4.3 Gesture for good

4.3 VHDL- Very high speed integrated circuit Hardware Description Language

VHDL stands for **V**HSIC (Very High Speed Integrated Circuits) **H**ardware **D**escription **L**anguage. It was developed with the goal to implement very high speed integrated circuits. It has now become one of the industries standard languages to design digital systems. It can also be used as a general purpose parallel programming language. It was originally developed at the U.S Department of defense. The IEEE standard 1076 defines VHDL. It declares all the data types like the numerical, logical, time, character etc.

The VHDL language can be regarded as an integrated amalgamation of the following languages:

Sequential language+ concurrent language+ net-list language+ timing specifications+ waveform generation language → VHDL.

It means that the language has constructs that enable us to express the concurrent or sequential behavior of a digital system with or without timing. It also allows us to model the system as an interconnection of components. Test waveforms can also be generated using the same constructs. The language not only defines the syntax but also defines very clear simulation semantics for each language construct. Therefore, models written in this language can be verified using a VHDL simulator. It is a strongly typed language and is often verbose to write.

4.3.1 Advantages of VHDL over Verilog

Verilog is another hardware description language that is commonly used in the research and industrial field. In our project we have selected the language as VHDL. Both languages are fundamentally the same with very less differences between them.

- ✓ VHDL is based on Pascal and Ada while Verilog is based on C.
- ✓ Unlike Verilog, VHDL is a strongly typed language.
- ✓ Verilog lacks library management unlike VHDL.
- ✓ VHDL is exact in its nature and hence very verbose. Verilog requires much more care to avoid nasty bugs.
- ✓ Verilog is intended as a simulation language.

- ✓ Verilog has only simple data types but VHDL allows us to create very complex data types as well.

4.3.2 Levels of representation and abstraction

There can be different levels of abstraction for a digital system. For VHDL the following figure shows the different levels of abstraction.

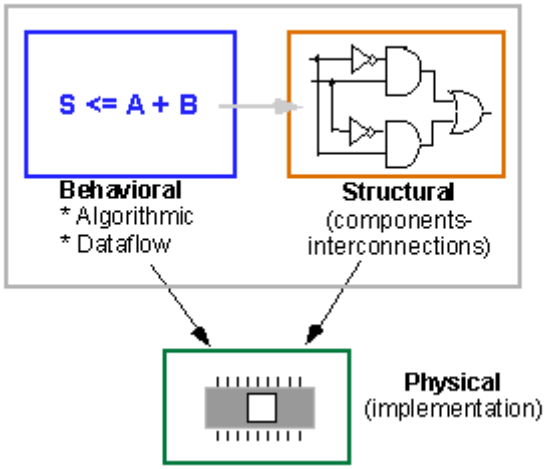

Fig. 4.4 Levels of abstraction

The highest level of abstraction is the behavioral level that describes a system in terms of what it does or how it behaves rather than in terms of its components and interconnection between them. A behavioral description specifies the relationship between the input and output signals. This could be a Boolean expression or more abstract description such as the register transfer or algorithmic level. The structural level, on the other hand, describes a system as a collection of gates and components that are interconnected to perform a desired function. A structural description could be compared to a schematic of interconnected logic gates. It is a representation that is usually closer to the physical realization of a system.

4.3.3 Basic structure of a VHDL file

A digital system in VHDL consists of a design entity that can contain other entities that are then considered components of the top-level entity. Each entity is modeled by an entity

declaration and an architecture body. One can consider the entity declaration as the interface to the outside world that defines the input and output signals, while the architecture body contains the description of the entity and is composed of interconnected entities, processes and components, all operating concurrently, as schematically shown in figure below:

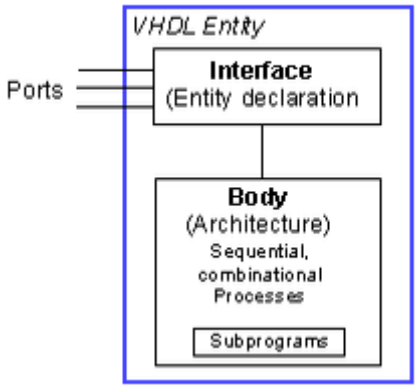

Fig. 4.5 A VHDL file

The fundamental units in VHDL are as follows:

1. LIBRARY declarations: Contains a list of all libraries to be used in the design. For example: ieee, std, work, etc.
2. ENTITY: Specifies the I/O pins of the circuit.
3. ARCHITECTURE: Contains the VHDL code proper, which describes how the circuit should behave (function).

An entity always starts with the keyword **entity**, followed by its name and the keyword **is**. Next are the port declarations using the keyword **port**. An entity declaration always ends with the keyword **end**, optionally [] followed by the name of the entity.

The general form is as follows,

```
entity NAME_OF_ENTITY is [ generic generic_declarations);]
     port (signal_names: mode type;
            signal_names: mode type;
                 :
            signal_names: mode type);
end [NAME_OF_ENTITY] ;
```

The architecture body specifies how the circuit operates and how it is implemented. As discussed earlier, an entity or circuit can be specified in a variety of ways, such as behavioral, structural (interconnected components), or a combination of the above.

The architecture body looks as follows,

```
architecture architecture_name of NAME_OF_ENTITY is
-- Declarations
      -- components declarations
      -- signal declarations
      -- constant declarations
      -- function declarations
      -- procedure declarations
      -- type declarations

      :

begin
-- Statements

      :

end architecture_name;
```

One of the major utilities of VHDL is that it allows the syn- thesis of a circuit or system in a programmable device (PLD or FPGA) or in an ASIC. The steps followed during such a project are summarized in the following figure:

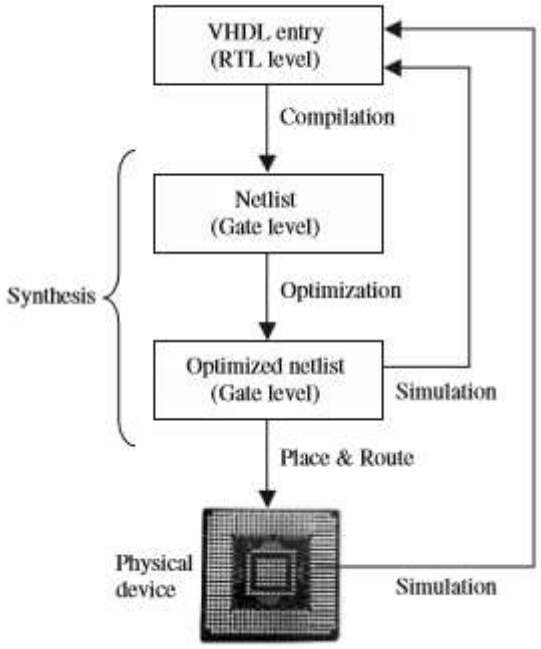

Fig. 4.6 Summary of VHDL design flow

There are several EDA (Electronic Design Automation) tools available for circuit synthesis, implementation, and simulation using VHDL. Some tools (place and route, for example) are offered as part of a vendor's design suite (e.g., Altera's Quartus II, which allows the synthesis of VHDL code onto Altera's CPLD/FPGA chips)

Thus the most important principle around which this project revolves is the VHDL implementation of all the modules that we need to realize this project.

5. Design

5.1 Block diagram

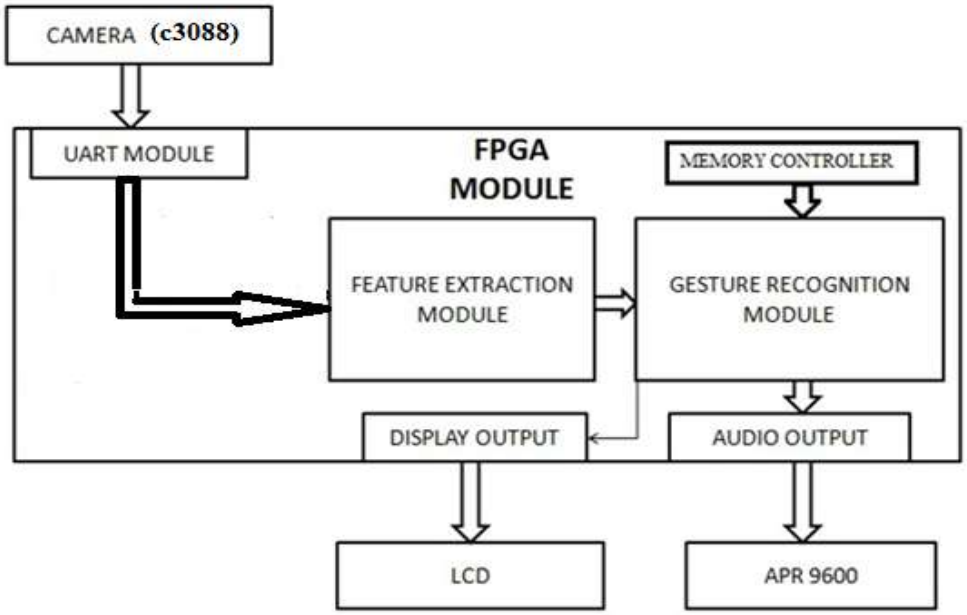

Fig 5.1 Block Diagram

5.2 Block description

This platform of this project is FPGA. FPGA is programmed to different modules for carrying out different functions.

5.2.1 FPGA

Field Programmable Gate Arrays (FPGAs) are semiconductor devices that are based around a matrix of configurable logic blocks (CLBs) connected via programmable interconnects. FPGAs can be reprogrammed to desired application or functionality requirements after manufacturing.

FPGAs contain programmable logic components called "logic blocks", and a hierarchy of reconfigurable interconnects that allow the blocks to be "wired together"—somewhat like many (changeable) logic gates that can be inter-wired in (many) different configurations. Logic blocks can be configured to perform complex combinational functions, or merely simple logic gates like AND and XOR. In most FPGAs, the logic blocks also include memory elements, which may be simple flip-flops or more complete blocks of memory.

The most common FPGA architecture consists of an array of logic blocks (called Configurable Logic Block, CLB, or Logic Array Block, LAB, depending on vendor), I/O pads, and routing channels. Generally, all the routing channels have the same width (number of wires). Multiple I/O pads may fit into the height of one row or the width of one column in the array.

An application circuit must be mapped into an FPGA with adequate resources. While the number of CLBs/LABs and I/Os required is easily determined from the design, the number of routing tracks needed may vary considerably even among designs with the same amount of logic. For example, a crossbar switch requires much more routing than a systolic array with the same gate count. Since unused routing tracks increase the cost (and decrease the performance) of the part without providing any benefit, FPGA manufacturers try to provide just enough tracks so that most designs that will fit in terms of Lookup tables (LUTs) and I/Os can be routed. This is determined by estimates such as those derived from Rent's rule or by experiments with existing designs.

In general, a logic block (CLB or LAB) consists of a few logical cells (called ALM, LE, Slice etc.). A typical cell consists of a 4-input LUT, a Full adder (FA) and a D-type flip-flop, as shown below. The LUTs are in this figure split into two 3-input LUTs. In normal mode those are

combined into a 4-input LUT through the left mux. In arithmetic mode, their outputs are fed to the FA. The selection of mode is programmed into the middle multiplexer. The output can be either synchronous or asynchronous, depending on the programming of the mux to the right, in the figure example. In practice, entire or parts of the FA are put as functions into the LUTs in order to save space.

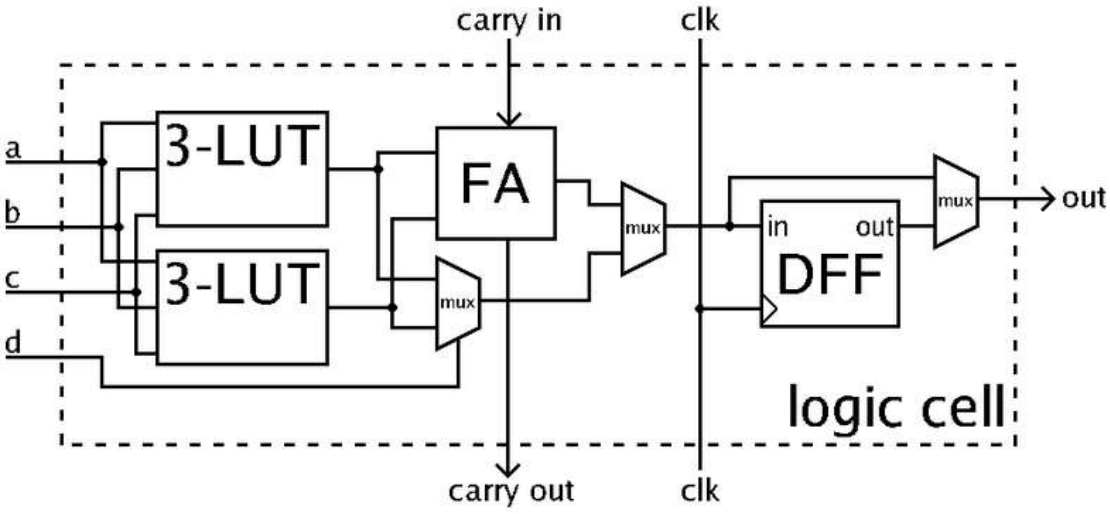

Fig 5.2 Architecture of FPGA

The FPGA Module intended to be used here for this specific application is ALTERA CYCLONE IV.

5.2.2 Camera (C3088)

The C3088 is a 1/4" color camera module with digital output. It uses Omni Vision's CMOS image sensor OV6620. Combining CMOS technology together with an easy to use digital interface makes C3088 a low cost solution for higher quality video image application. The digital video port supplies a continuous 8/16 bit-wide image data stream. All camera functions, such as exposure, gamma, gain, white balance, color matrix, windowing, are programmable through I2C interface. In combine with OV511+, USB controller chip, it will be easily form a USB camera for PC application.

5.2.3 UART Module

Universal Asynchronous Receiver Transmitter (UART) is a kind of serial communication protocol; mostly used for short-distance, low speed, low-cost data exchange between computer and peripherals. UARTs are used for asynchronous serial data communication by converting data from parallel to serial at transmitter with some extra overhead bits using shift register and vice versa at receiver. Serial communication reduces the distortion of a signal, therefore makes data transfer between two systems separated in great distance possible. It is generally connected between a processor and a peripheral, to the processor the UART appears as an 8-bit read/write parallel port. The UART implemented with VHDL language can be integrated into the FPGA to achieve compact, stable and reliable data transmission.

Basic UART communication needs only two signal lines (RXD, TXD) to complete full-duplex data communication. TXD is the transmit side, the output of UART; RXD is the receiver, the input of UART. UART‟s basic features are: There are two states in the signal line, using logic 1 (high) and logic 0 (low) to distinguish respectively. For example, when the transmitter is idle, the data line is in the high logic state. Otherwise when a word is given to the UART for asynchronous transmissions, a bit called the "Start Bit" is added to the beginning of each word that is to be transmitted. The Start Bit is used to alert the receiver that a word of data is about to be sent, and to force the clock in the receiver into synchronization with the clock in the transmitter. After the Start Bit, the individual data bits of the word are sent, with the Least Significant Bit (LSB) being sent first. Each bit in the transmission is transmitted for exactly the same amount of time as all of the other bits, and the receiver “looks” at the wire at approximately halfway through the period assigned to each bit to determine if the bit is a 1 or a 0. When the entire data word has been sent, the transmitter may add a Parity Bit that the transmitter generates.

The Parity Bit may be used by the receiver to perform simple error checking. Then at least one Stop Bit is sent by the transmitter. When the receiver has received all of the bits in the data word, it may check for the Parity Bits (both sender and receiver must agree on whether a Parity Bit is to be used), and then the receiver looks for a Stop Bit. Regardless of whether the data was received correctly or not, the UART automatically discards the Start, Parity and Stop bits. If the sender and receiver are configured identically, these bits are not passed to the host. If another word is ready for transmission, the Start Bit for the new word can be sent as soon as the

Stop Bit for the previous word has been sent. Because asynchronous data are “self synchronizing”, if there are no data to transmit, the transmission line can be idle. The UART frame format is shown in Fig. 5.3.

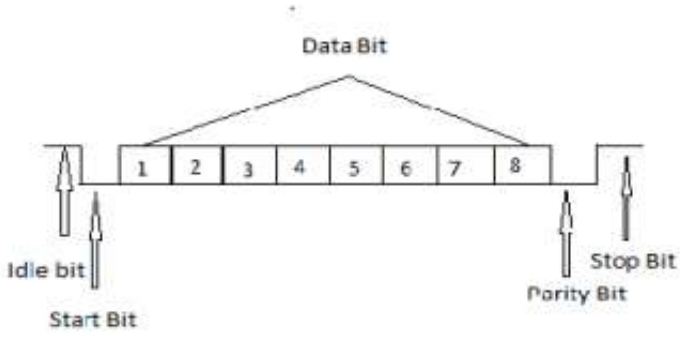

Fig 5.3 UART Frame Format

In this project, we use UART which includes three modules which are the baud rate generator, receiver and transmitter. The proposed design of UART satisfies the system requirements of high integration, stabilization, low bit error rate, and low cost. It also supports configurable baud rate generator with data length of 8 bits per frame. The configurable baud rate generator has been implemented using two switches of cyclone II FPGA.

5.2.4 Feature extraction module

Extraction of unique features for each gesture which are independent of human hand size and light illumination is important. Shape based recognition requires very less computation effort and saves a lot of FPGA area as compared to other approaches. But the approach is not very popular as a hand can assume many shapes. So, in the proposed hand gesture recognition system four different shape based features are calculated rather than relying on a single feature. The extracted features are then compared with the stored features of different gestures.

5.2.5 Memory controller

This module stores the previously extracted features of standard gestures. The features extracted from the image of gestures are compared with these data to categorize the gestures.

FPGAs have no standard memory hierarchy. They contain lots of memory blocks which have to be combined to build a custom memory system. There are many similarities with scratch-pad memory based systems: user-defined explicit data transfers, burst transfers (cf. pre fetch) between (external and internal) memories in parallel with computations.

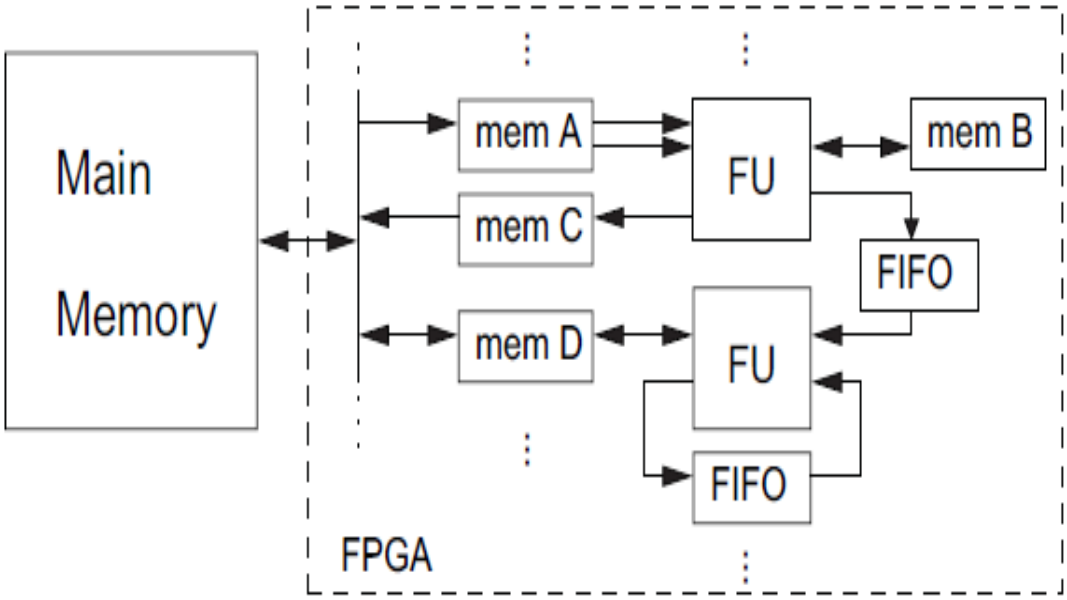

Fig 5.4 Memory hierarchy on an FPGA (FU = Functional Unit)

5.2.6 Gesture recognition module

This module compares the stored features of gestures with the features extracted from the real time images. If they match the gesture is recognized and corresponding signals are sent to the display and audio output of the FPGA.

5.2.7 APR9600

The APR9600 device offers true single-chip voice recording, non-volatile storage, and playback capability for 40 to 60 seconds. The device supports both random and sequential access of multiple messages. Sample rates are user-selectable, allowing designers to customize their design for unique quality and storage time needs. Integrated output amplifier, microphone amplifier, and AGC circuits greatly simplify system design.

APR9600 is used in this project for recording the voice messages corresponding to the gestures and playing back them according to the commands from the FPGA.

5.2.8 LCD Display

LCD display is used for displaying the message corresponding to the gesture for the deaf people. The messages stored in the LCD will be displayed corresponding to its input from the FPGA.

6. Implementation

The image captured from the camera is given to the edge detection module to convert the image into more distinguishable form.

6.1 Image Acquisition

The image of hand for processing is acquired using CMOS image sensor camera, which generates the R, G, B components for the image. The mathematical model of image acquisition process is represented as

$$R = \sigma \int E(\lambda) S(\lambda) Q_R(\lambda) d\lambda$$

$$G = \sigma \int E(\lambda) S(\lambda) Q_G(\lambda) d\lambda$$

$$B = \sigma \int E(\lambda) S(\lambda) Q_B(\lambda) d\lambda \qquad (6.1)$$

where σ denotes a constant factor, $E(\lambda)$ is spectral distribution, $s(\lambda)$ is the surface reflectance of the hand and other objects in the region and QR(λ) , QB (λ), QG(λ) are spectral sensitivities of the CMOS image sensor.

The captured image is retrieved by the camera interfacing module in R, G, B components. The camera interfacing module is designed to setup communication between camera and the FPGA board. The image is then stored on board DDR RAM and magnified to the 320×240 pixels resolution around the region of interest (ROI) as the resolution of the camera may be set to different value. The resolution of the camera may be adjusted manually. The distance between the camera and hand is about 35 centimeters and the hand gesture is held in predefined region of interest to avoid invariance due to location of hand with respect to camera. The FPGA is operated at 50 MHz, which allows acquiring the images in real time.

6.2 Gesture recognition

Human hands can show different gestures depending upon the shape of the hand pattern. The different hand gestures can be distinguished on the basis of shape based features. In this section, four different shape based features viz. area, perimeter, thumb detection and radial profile of hand are described; and the proposed criterion for classification for hand gestures is defined.

6.2.1 Features extraction

Shape based recognition requires very less computation effort and saves a lot of FPGA area as compared to other approaches. But the approach is not very popular as a hand can assume many shapes. So, in the proposed hand gesture recognition system four different shape based features are calculated rather than relying on a single feature. The different shape based features used in the proposed system are described as follows

6.2.1.1 Area of hand

Hand can occupy different areas for different hand gestures. So, the different gestures can be differentiated on the basis of area. The areas of the hand gestures are computed in accordance with

$$Area\ of\ Hand = \sum_{x=0}^{240} \sum_{y=0}^{320} I(x,y)(segmented) \tag{6.2}$$

.

6.3 Flow chart

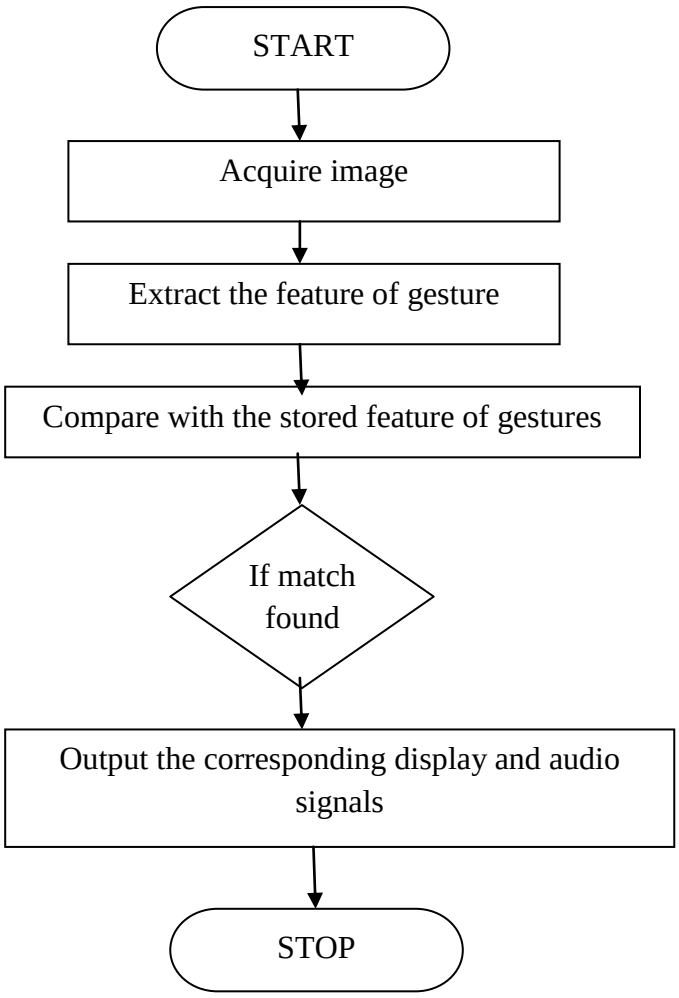

Fig 6.1 Flow chart of gesture recognition

6.4APR9600 for recording and playback of single message

6.4.1 Functional Description

APR9600 block diagram is included in order to describe the device's internal architecture. At the left hand side of the diagram are the analog inputs. A differential microphone amplifier, including integrated AGC, is included on-chip for applications requiring use. The amplified microphone signals fed into the device by connecting the ANA_OUT pin to the ANA_IN pin through an external DC blocking capacitor. Recording can be fed directly into the ANA_IN pin through a DC blocking capacitor, however, the connection between ANA_IN and ANA_OUT is still required for playback. The next block encountered by the input signal is the internal anti-aliasing filter. The filter automatically adjusts its response according to the sampling frequency selected so Shannon's Sampling Theorem is satisfied.

After anti-aliasing filtering is accomplished the signal is ready to be clocked into the memory array. This storage is accomplished through a combination of the Sample and Hold circuit and the Analog Write/Read circuit. These circuits are clocked by either the Internal Oscillator or an external clock source.

When playback is desired the previously stored recording is retrieved from memory, low pass filtered, and amplified as shown on the right hand side of the diagram. The signal can be heard by connecting a speaker to the SP+ and SP- pins. Chip-wide management is accomplished through the device control block shown in the upper right hand corner. Message management is provided through the message control block represented in the lower center of the block diagram.

More detail on actual device application can be found in the Sample Application section. More detail on sampling control can be found in the Sample Rate and Voice Quality section. More detail on Message management and device control can be found in the Message Management section.

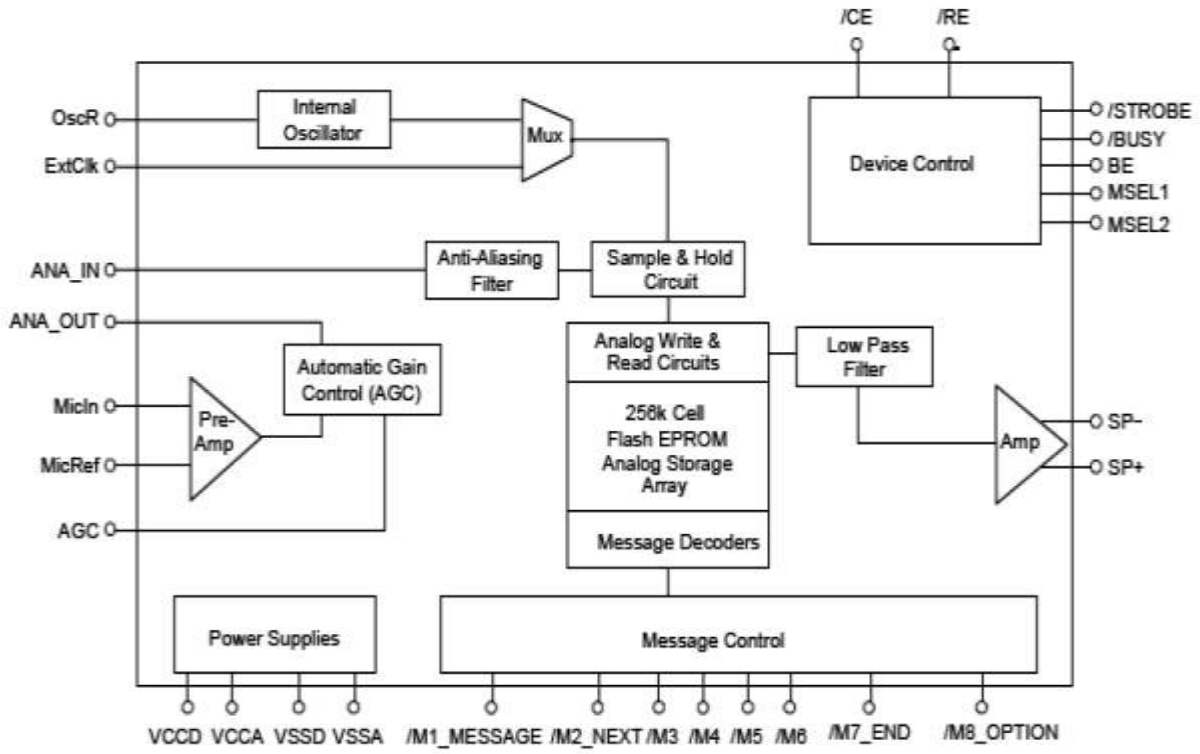

Fig 6.2 Block diagram of APR9600

Recording mode:

On power-up, the device is ready to record or playback, starting at the first address in the memory array. Before you can begin recording, the /CE input must be set to low to enable the device and /RE must be set to low to enable recording. On a falling edge of the /M1_MESSAGE pin the device will beep once and initiate recording. A subsequent rising edge on the /M1 Message pin will stop recording and insert a single beep. If the M1_MESSAGE pin is held low beyond the end of the available memory, recording Stops automatically, and two beeps are inserted; regardless of the state of the /M1_MESSAGE pin. The device returns to the standby mode when the /M1_MESSAGE pin is returned high. A subsequent falling edge on the /M1_MESSAGE pin starts a new record operation in the memory array immediately following the last recorded message, thus preserving the last recorded message. To record over all previous messages you must pulse the /CE pin low once to reset the device to the beginning of the first message.

Playback mode:

On power-up, or after a low to high transition on /RE the device is ready to record or playback starting at the first address in the memory array. Before you can begin playback of messages, the /CE input must be set to low to enable the device and /RE must be set to high to enable playback. The first high to low going pulse of the /M1_MESSAGE pin initiates playback from the beginning of the current message. When the /M1_MESSAGE pin pulses from high to low a second time, playback of the current message stops immediately.

When the/M1_MESSAGE pin pulses from high to low a third time, playback of the next message starts again from the beginning. If you hold the /M1_MESSAGE pin low continuously, the current message and subsequent messages play until the one of the following conditions is met: the end of the memory array is reached, the last message is reached, The /M1_message pin is released.

Table 1

Mode	MSEL1	MSEL2	/M8_OPTION
Random Access 2 fixed duration messages	0	1	Pull this pin to VCC through I00K resistor
Random Access 4 fixed duration messages	1	0	Pull this pin to VCC through I00K resistor
Random Access 8 fixed duration messages	1	1	The /M8 message trigger becomes input pin
Tape mode, Auto rewind operation	0	0	0
Tape mode, Normal operation	0	0	1

Table 6.1. Different modes

6.5 APR9600 circuit

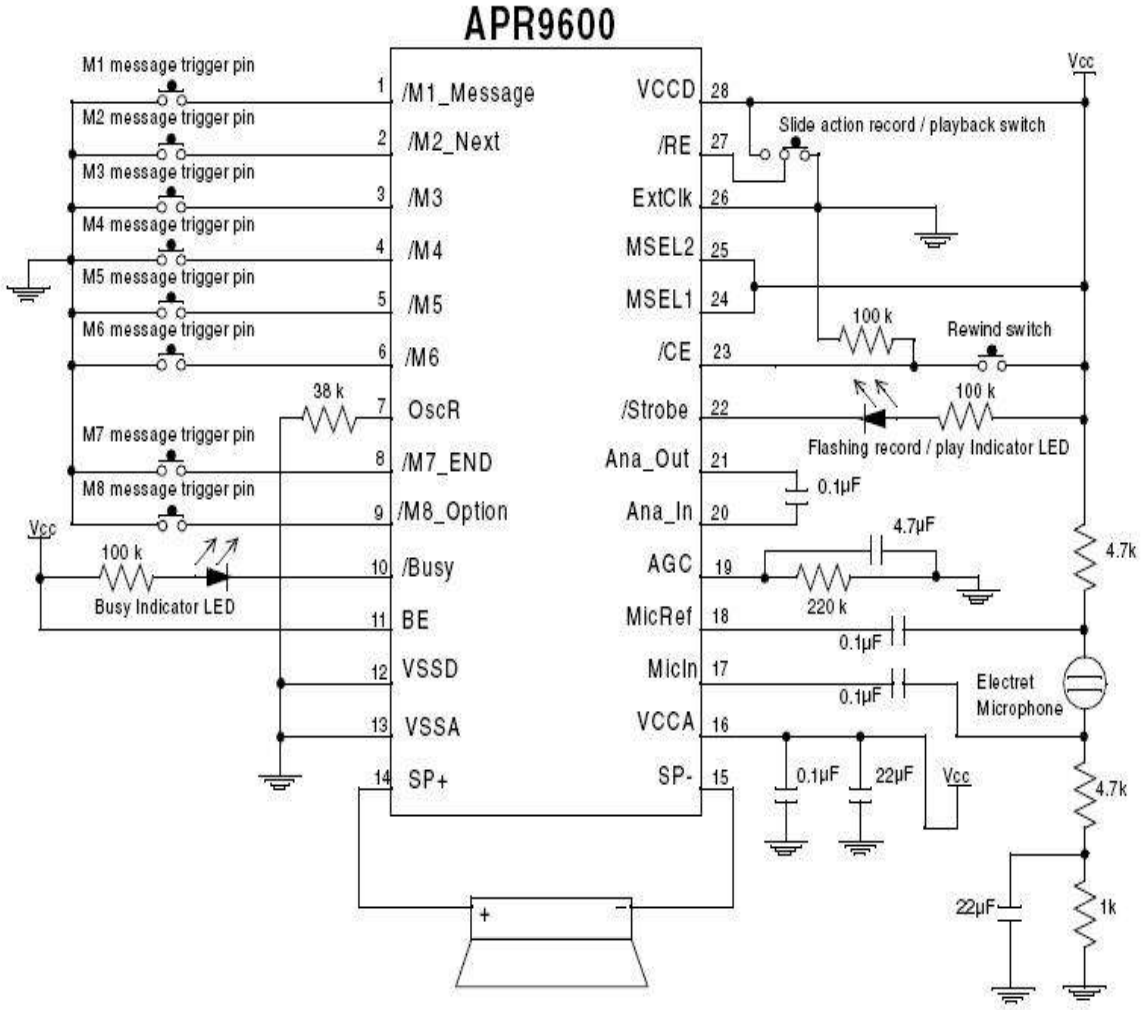

Fig. 6.3 Circuit diagram of APR9600

6.5.1Circuit Description

- The proposed circuit of a programmable single chip voice recorder/player utilizes the IC APR9600 as the main processor of the circuit.
- It is a 28 pin IC which can be very easily and quickly configured for getting the required results by adding a handful of common passive electronic components.
- All the pin outs of the IC are specified by their individual functions, and the components are accordingly attached with the respective pin outs. For example pin#28 and pin#27 are assigned as the trigger inputs for initiating playback and recording functions.
- Sliding the connected switch toward right initiates the playback action while toggling it toward left puts the IC in the recording mode.
- The IC also has appropriate visual indication options which provide the user with instant information regarding the position of the circuit.
- The LED at pin#8 indicates the end of a playback file session.
- The LED at pin#10 stays illuminated for so long the audio is being played, indicating circuit "busy"
- The LED at pin#22 indicates through rapid flashes regarding the playback or recording modes of the IC.
- The input data is normally picked from the mic which is appropriately connected across the pins 17 and 18 of the IC.
- When the slider switch is pushed toward the recording mode, any audio entering the mic gets stored inside the IC until the specified time elapses.
- The sampling rate of the IC can be set as per the user preference. Lower sampling rates will provide longer recording/playback periods and vice versa.
- Longer periods would also mean lower voice quality while shorter periods of recording spec will produce relatively better sound processing and storing.
- The entire circuit operates with a 5 volt supply which can be acquire through a standard 7805 IC after rectification from a standard transformer bridge capacitor network.
- The audio output may be derived across pin#14 and ground which must be terminated to an audio amplifier so that the data can be heard with proper volume.

6.6 Algorithms

6.6.1 Baud generation

1) Start the program.
2) Include the necessary library files.
3) Create an **entity** named “baud_gen” which includes two input ports (clk and baud reset) and one output port (baud).
4) End the entity.
5) Define the **architecture** of the entity.
6) Define two signals named “cnt” and “blink” also define a constant named “cnt_max”(215).
7) Begin the architecture with a **process** dependant on the two input ports.
8) For each rising edge of the clock check for the value of baud reset. if it is “1” then initialize “cnt” and ‘blink’ to ‘0’.
9) Else check the value of ‘cnt’ .if it is equal to ‘cnt_max’ then make ‘cnt =0’ and invert the value of ‘blink’, otherwise increment the value of ‘cnt’ by ‘1’.
10) End the process
11) Assign the value of blink to baud.
12) End the architecture.

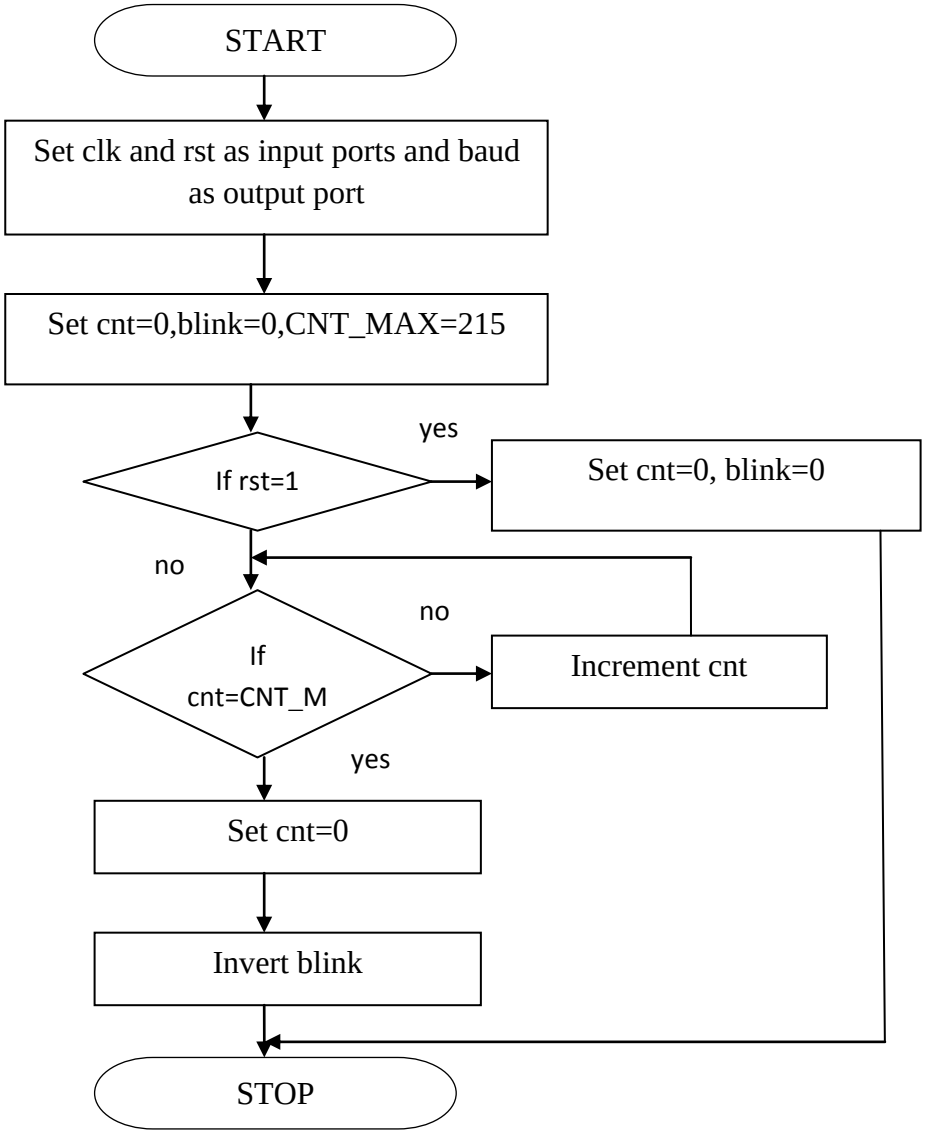

Fig 6.4 Flow chart of baud generation

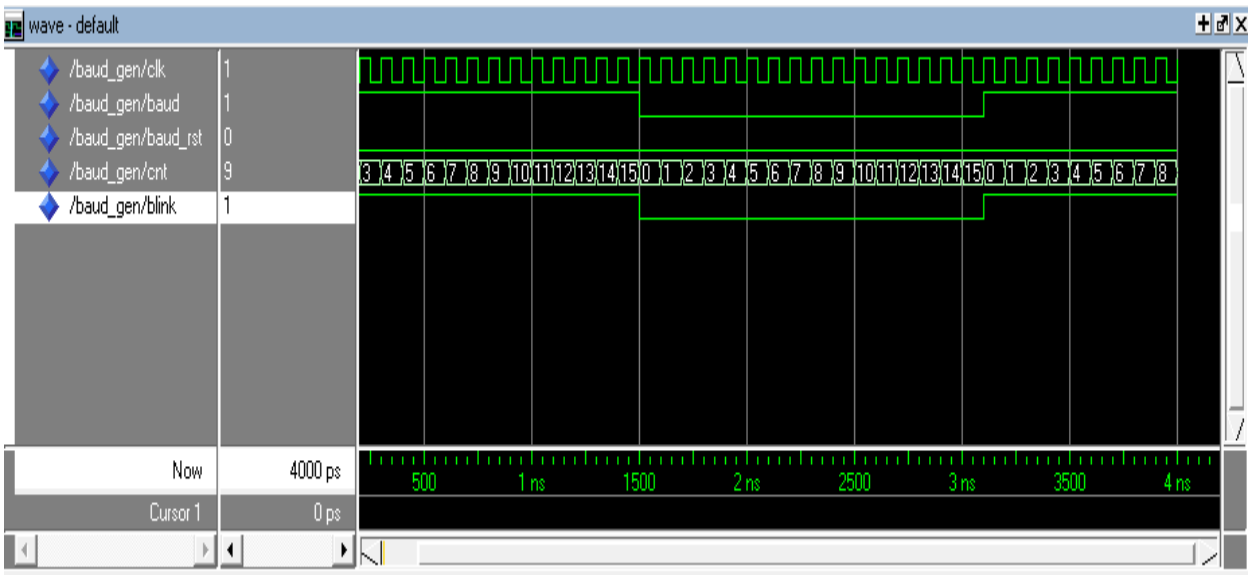

Fig.6.5 Simulation of baud generation

6.6.2 UART transmission

1) Start the program.
2) Include the necessary library files.
3) Create the entity UART_TX which includes input ports ‘baud’, ‘clk’, ‘rst’, ‘txen’, ‘tbuff’and the output ports trmt and tx.
4) End the entity.
5) Define the architecture of the entity.
6) Define the type named uart_states which includes the bits ideal, start, tx0, tx1, tx2, tx3, tx4,tx5,tx6,stop1,stop2,stop3.
7) Define the signal tx_next as uart_states.
8) Begin the architecture with the process which includes ‘rst’, ‘baud’ and ‘clk’.
9) If ‘rst=1’ then assign ideal state to tx_next. also make tx and trmt ports logic high.
10) For each rising edge of the baud already created by the baud_gen programme check for the value of txen.
11) If it is ‘1’ then continuously check the value of tx_next using switch case.
12) Since tx_next is idle, asiign logic high to tx and trmt and state start to tx_next.
13) Now since the state is switched to start assign ‘0’ to tx port and tx0 to tx_next.

14) The state now switches to tx0 where tbuff(0) is assigned to tx port and tx_next is again switched to tx1.
15) This continues until tx_next state again becomes idle.
16) End the process.
17) End the architecture.

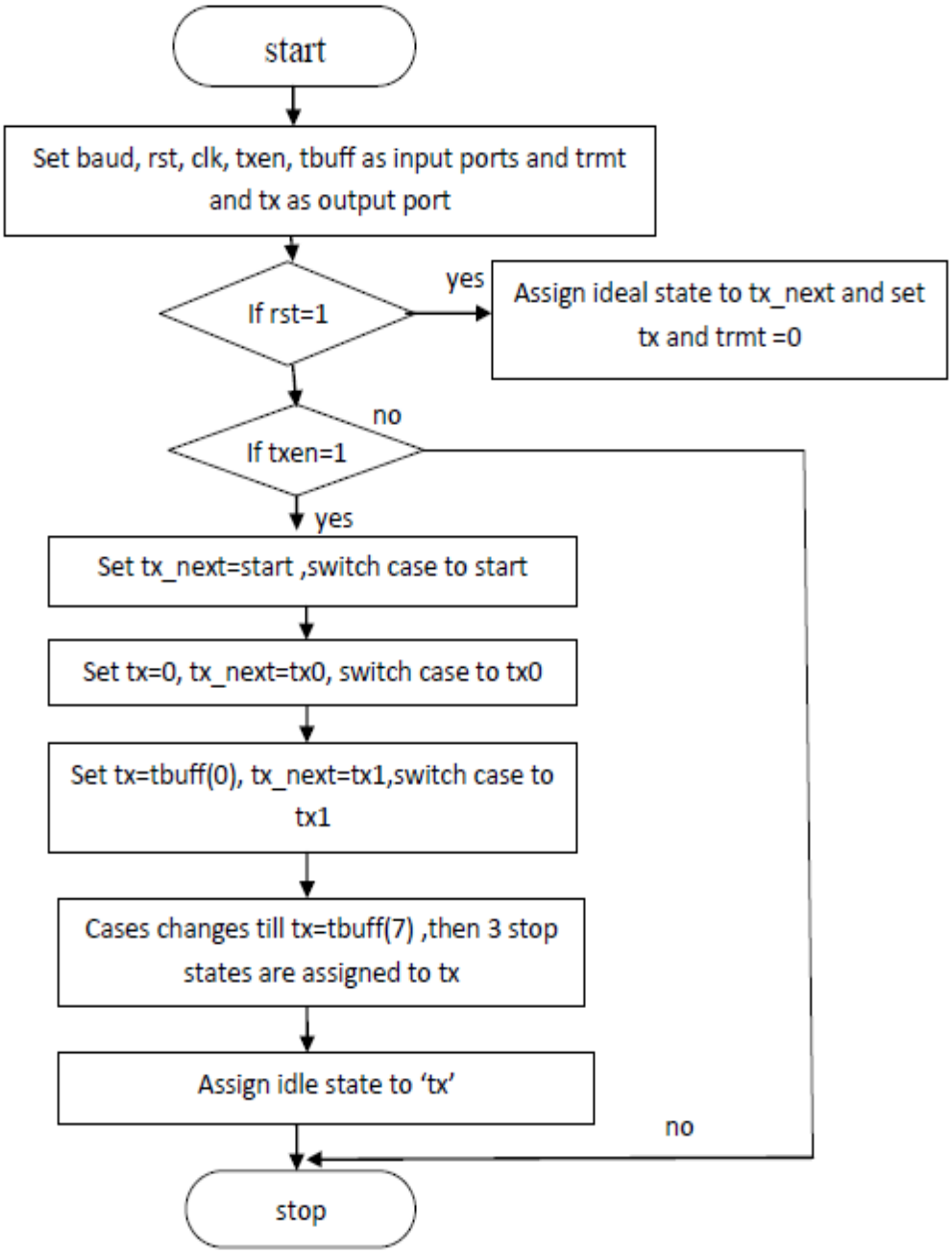

Fig 6.6 Flow chart of uart transmitor

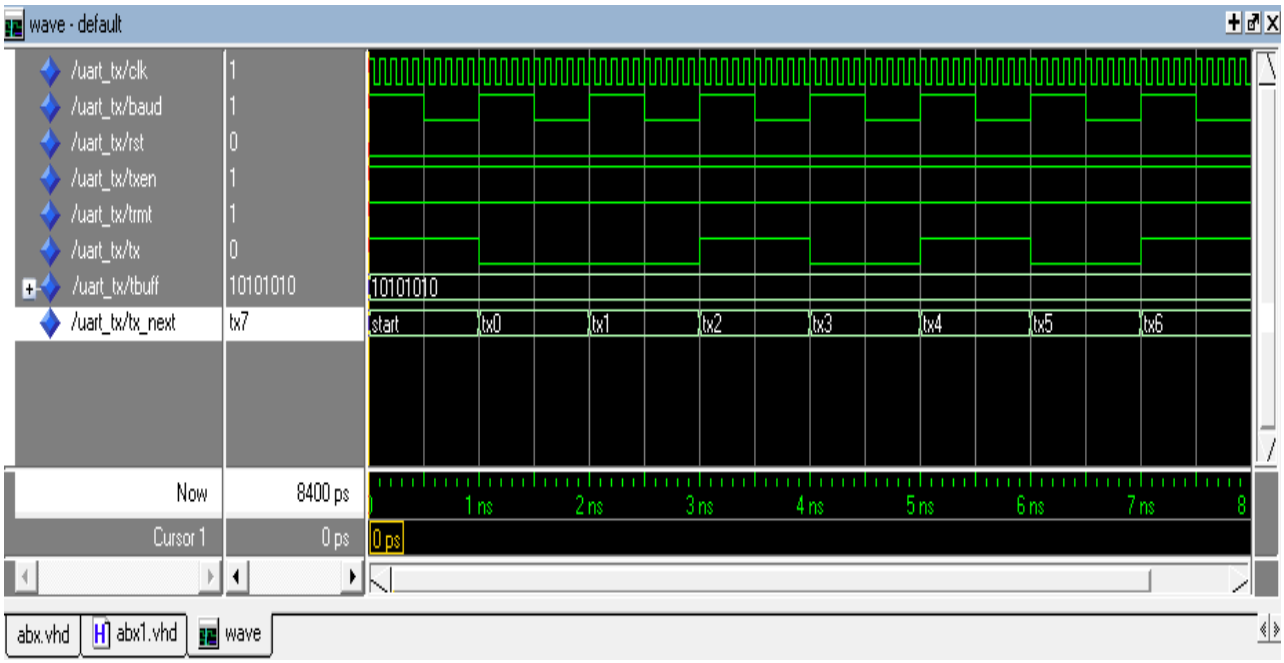

Fig 6.7 Simulation of uart transmitter

6.6.3 LCD Display

1) Start the program.
2) Include the necessary library files.
3) Create the entity named ‘lcd_controller’ which includes input ports ‘clk’,’rst’ and ‘lcd_if’ and output ports lcd_dat, lcd_on, lcd_cmt, lcd_init and lcd_cmd.
4) End the entity.
5) Define the architecture of the entity.
6) Define the type named ascii_tx_strings as a vector of 8 bits.
7) Assign the signal ascii_tx_string to the type defined in the earlier step.
8) Define ascii value for every alphabets using constant. ‘A’ is assigned ‘41’ and ‘Z’ is assigned ‘5A’.
9) Define the type named ctrl_states which includes start,state1, busy, busy1 and stop.
10) Assign the signal present_state to the type ctrl_states.
11) Define the ascii_tx_string positions with the required alphabets to be displayed.
12) Begin the process which is based on clk.

13) For each rising edge of the clock check for the value of ‘rst’. If it is ‘0’ then initialise all the input ports.
14) If it is ‘’1’ then using switch case check the value of present_state.
15) For present_state equal to start make the char_count value ‘0’ and switch the control to busy state.
16) From there switch to state1 wherein the stored ascii_tx_string is is assigned to lcd_cd.
17) Increment the value of char_count.
18) Continue it until the value of char_count equals the no.of alphabets to be displayed.
19) End the process.
20) End the architecture.

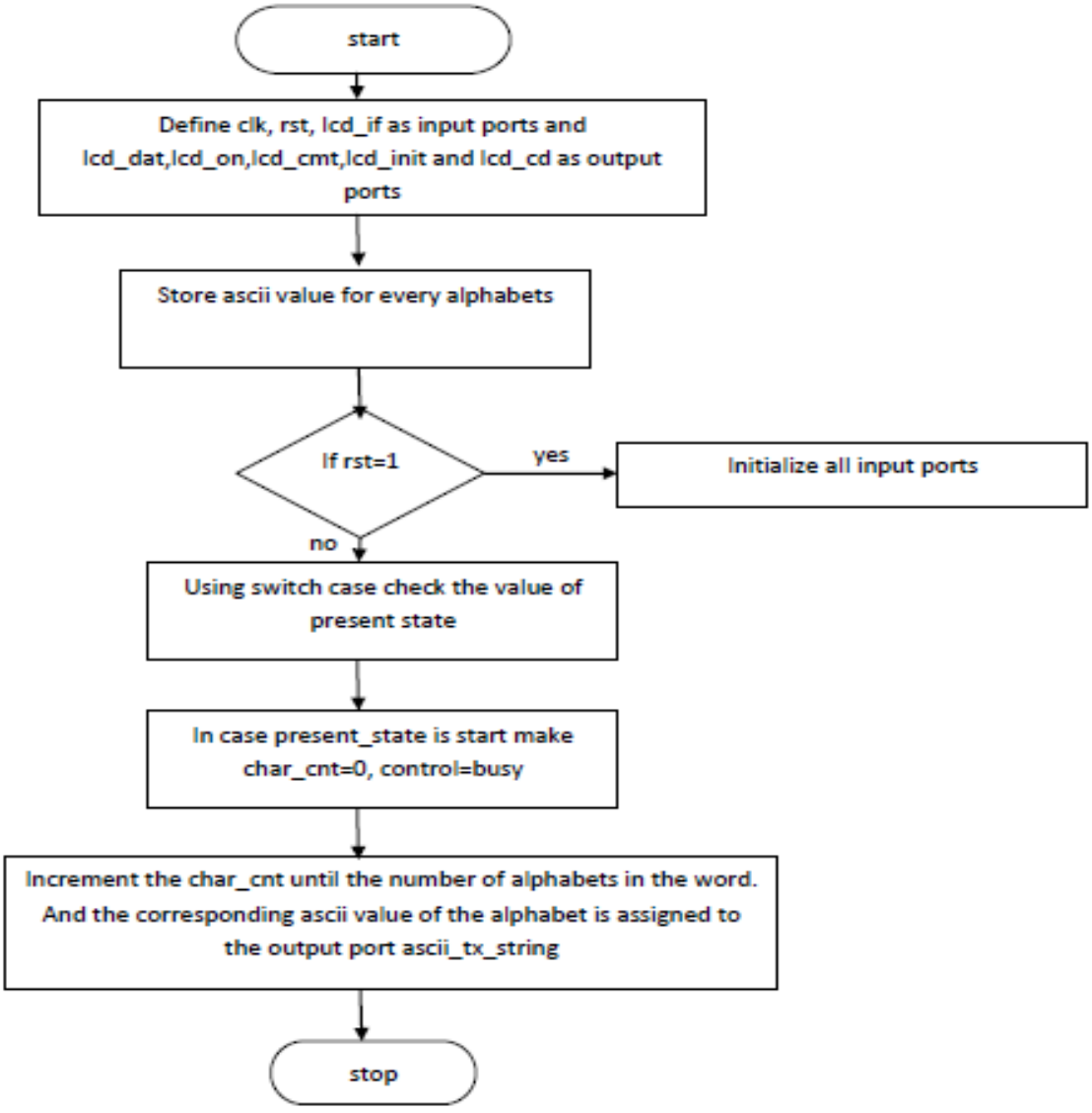

Fig 6.8 Flow chart of lcd

6.6.4 Camera controller

1) Start the program
2) Include the necessary files
3) Include a port named cam to the earlier designed UART controller. it has CAM_BUS, PIC_RDY, DOK, QFLAG as inputs. CAP_PIC and DIF are outputs.
4) UART_TX sends READ to MATLAB inorder to convey that UART is ready.
5) UART waits for receiving character 'C' (0x43).
6) UART_RX constantly checks for data in RBUFF provided RXIF makes a transition from logic 1 to 0.
7) If RBUFF = "43" ie, 'C' then CAP_PIC is made 1.
8) After receiving CAP_PIC the system waits for the rising edge of vsync
9) After receiving vsync, the system waits to receive the rising edge of href and then for rising edge of pclk.
10) Camera starts to receive data. First data received is green. We avoid green.
11) We wait for red pixel in the next pclk. We store the red pixels in cpix_buff.
12) On receiving DIF from uart, camera sends 176*288 datas to uart controller. PIC_RDY is made high at this time.
13) QFLAG is set high after entire data is sent to uart controller.
14) DIF and CAP_PIC is made logic 0 and uart again waits for character 'C' to be received from MATLAB.

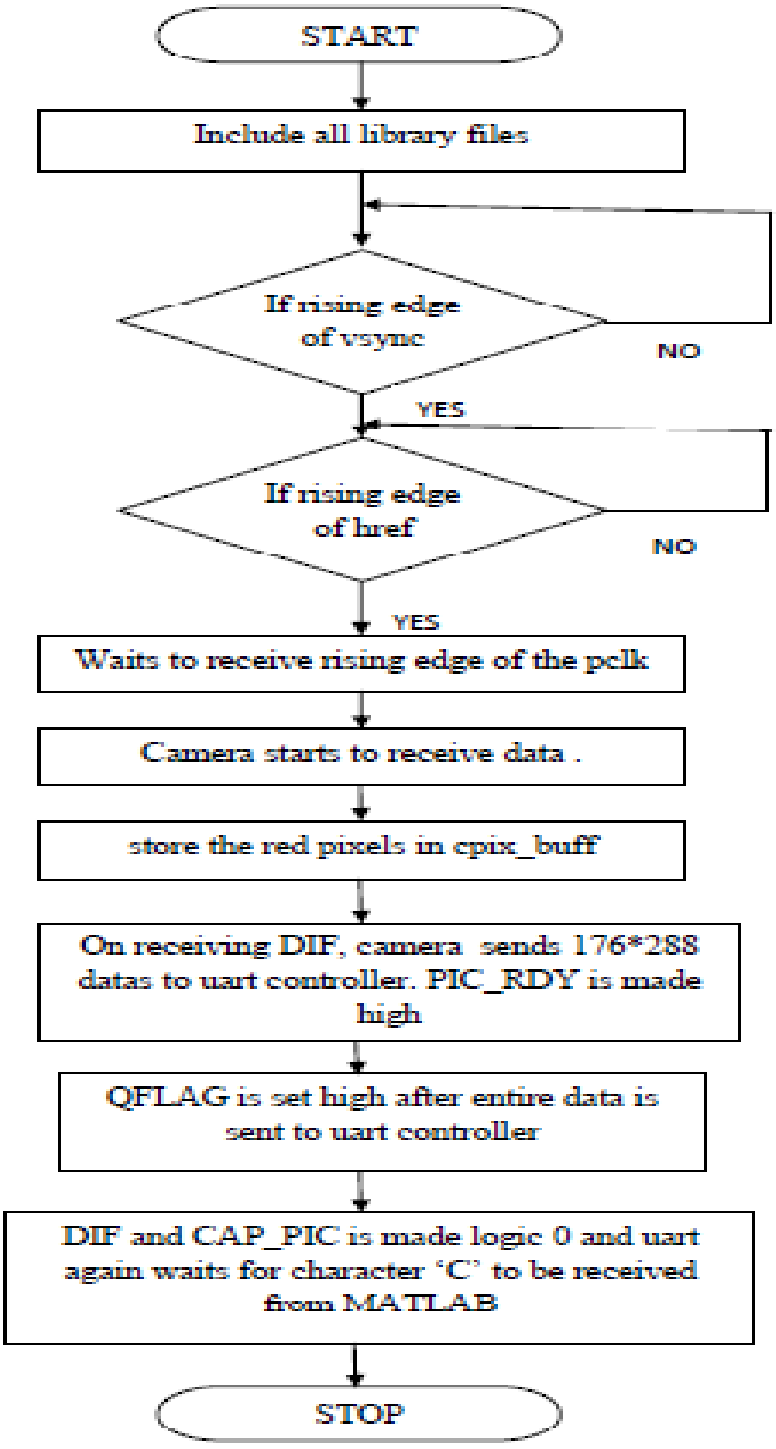

Fig 6.9 Flow chart of camera controller

Appendix A

A.1 Program for baud generation

```
library ieee;
 use ieee.std_logic_1164.all;
 use ieee.std_logic_unsigned.all;

entity BAUD_GEN is

port (
   CLK     : in std_logic;
   BAUD     : out std_logic;
   BAUD_RST  : in std_logic
);
end BAUD_GEN;

architecture ABAUD_GEN of BAUD_GEN is

   constant CNT_MAX  : integer  range 0 to 255 := 215;

   signal cnt   : integer  range 0 to 255;
   signal blink    : std_logic := '1';

begin

   process(CLK,BAUD_RST)
   begin

      if (CLK' event and CLK = '1') then
      if(BAUD_RST = '1')then cnt <= 0;blink <= '0';
      else
         if cnt = CNT_MAX then
            cnt <= 0;
            blink <= not blink;

         else
            cnt <= cnt + 1;
         end if;
      end if;
 end if;
      end process;
BAUD <= blink;


end ABAUD_GEN;
```

A.2 Program for UART controller

```
library ieee;
use ieee.std_logic_1164.all;
--use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
--use work.ASCII_pkg.all;

entity UART_Controller is
    port(
        RST            :   in std_logic;
        CLK            :   in std_logic;
        BAUD            :   in std_logic;
        -- TX
        TXEN           : out  std_logic;
        TBUFF          : out  std_logic_vector (7 downto 0);
        TRMT           : in   std_logic;

        -- RX
        CREN           : out  std_logic;
        RXIF           : in std_logic;
        RBUFF          : in std_logic_vector (7 downto 0);
           BAUD_RST       : out  std_logic;
        -- cam
        CAM_BUS      : in std_logic_vector (7 downto 0);
        CAP_PIC         : out  std_logic;
        PIC_RDY      : in  std_logic;
        DIF             : out  std_logic;
        DOK          : in  std_logic ;
        QFLAG          : in  std_logic

       );




end UART_Controller;

architecture UART_Controller_A of UART_Controller is

type ascii_tx_strings is array (integer range 0 to 5) of  std_logic_vector
(7 downto 0);

signal ascii_tx_string : ascii_tx_strings;



type ctrl_states is
(ideal,state0,state1,state_r1,state_r2,state_p1,state_p2,state_p3,state_tx
,busy1,busy2,busy3,busy);
signal  next_state      : ctrl_states ;
```

```
--signal  rtn_state        : ctrl_states ;

signal char_count : integer range 0 to 7;
signal tx_count : integer range 0 to 7;

signal update : std_logic;
signal RBUFF_temp1 : std_logic_vector (7 downto 0);
signal RBUFF_temp2 : std_logic_vector (7 downto 0);
signal tx_data  : std_logic_vector (7 downto 0);



begin

ascii_tx_string(0)  <= x"52";
ascii_tx_string(1)  <= x"45";
ascii_tx_string(2)  <= x"41";
ascii_tx_string(3)  <= x"44";



p1:process(CLK)
begin

if(CLK'event and CLK = '1') then

    if (RST = '1' ) then
       TXEN <= '0';
       TBUFF    <= "00000000";
       RBUFF_temp1 <= "00000000";
       RBUFF_temp2 <= "00000000";
       CREN     <= '0';
       next_state <= ideal;
       char_count <= 0;

    else

    case next_state is
        when busy =>
            if(TRMT = '0')then
                next_state <= state1;
                BAUD_RST <= '1';
                TXEN <= '0';
            end if;

        when busy1 =>
            if(TRMT = '1')then
                next_state <= busy;
            end if;

 ------------------------------------------------
        when busy3 =>
            if(TRMT = '0')then
                       next_state <= state_p2;
```

```
                    BAUD_RST <= '0';
                    TXEN <= '0';
            end if;

         when busy2 =>
            if(TRMT = '1')then
                next_state <= busy3;
            end if;

-----------------------------------------------------



        when state_tx =>

                 if(DOK = '1')then
                  DIF <= '0';
              TBUFF <= CAM_BUS;
              TXEN <= '1';
              next_state <= busy2;
              CAP_PIC <= '0';
                 end if;

        when state_p2 =>
            if(QFLAG = '1')then
                        DIF <= '0';
                        CAP_PIC <= '0';
                        next_state <= state_r1;
                        CREN <= '1';
                        BAUD_RST <= '0';
             elsif(PIC_RDY = '1') then
                        DIF <= '1';
                        CAP_PIC <= '0';
                 next_state <= state_tx;
                 CREN <= '0';
            end if;

         when state_p1 =>
            if(BAUD = '1' and RXIF = '0') then
                        if(RBUFF = x"43")then
                             next_state <= state_p2;
                             CAP_PIC <= '1';
                 end if;
            end if;


         when state_r1 =>
            if(BAUD = '1' and RXIF = '1') then
               next_state <= state_p1;
            end if;

        when state1 =>
        if(char_count <= 3) then
```

```
            TBUFF <= ascii_tx_string(char_count);
            char_count  <= char_count + 1;
            TXEN <= '1';
            next_state <= busy1;

        else
            next_state <= state_r1;
            CREN <= '1';
            BAUD_RST <= '0';
        end if;


        when ideal => char_count <= 0;
                  tx_count <= 0;
            next_state <= state1;

        when others => null;

    end case;
end if;
end if;
end process;
end UART_Controller_A;
```

A.3 Program for LCD display

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
use work.ASCII_pkg.all

entity lcd_Controller is
    port(

        CLK       : in std_logic;
        RST       : in std_logic;

             LCD_IF    : in std_logic;
             LCD_INIT  : out std_logic;
             LCD_CMD   : out std_logic;
             LCD_DAT   : out std_logic;
        LCD_ON    : out std_logic;
        LCD_CD    : out std_logic_vector(7 downto 0)
       );




end lcd_Controller;

architecture lcd_Controller_A of lcd_Controller is


type ascii_tx_strings is array (integer range 0 to 30) of
std_logic_vector (7 downto 0);

signal ascii_tx_string : ascii_tx_strings;

constant ccA  : std_logic_vector(7 downto 0) := x"41";
constant ccB  : std_logic_vector(7 downto 0) := x"42";
constant ccC  : std_logic_vector(7 downto 0) := x"43";
constant ccD  : std_logic_vector(7 downto 0) := x"44";
constant ccE  : std_logic_vector(7 downto 0) := x"45";
constant ccF  : std_logic_vector(7 downto 0) := x"46";
constant ccG  : std_logic_vector(7 downto 0) := x"47";
constant ccH  : std_logic_vector(7 downto 0) := x"48";
constant ccI  : std_logic_vector(7 downto 0) := x"49";
constant ccJ  : std_logic_vector(7 downto 0) := x"4A";
constant ccK  : std_logic_vector(7 downto 0) := x"4B";
constant ccL  : std_logic_vector(7 downto 0) := x"4C";
constant ccM  : std_logic_vector(7 downto 0) := x"4D";
constant ccN  : std_logic_vector(7 downto 0) := x"4E";
constant ccO  : std_logic_vector(7 downto 0) := x"4F";
constant ccP  : std_logic_vector(7 downto 0) := x"50";
constant ccQ  : std_logic_vector(7 downto 0) := x"51";
constant ccR  : std_logic_vector(7 downto 0) := x"52";
```

```
constant ccS  : std_logic_vector(7 downto 0) := x"53";
constant ccT  : std_logic_vector(7 downto 0) := x"54";
constant ccU  : std_logic_vector(7 downto 0) := x"55";
constant ccV  : std_logic_vector(7 downto 0) := x"56";
constant ccW  : std_logic_vector(7 downto 0) := x"57";
constant ccX  : std_logic_vector(7 downto 0) := x"58";
constant ccY  : std_logic_vector(7 downto 0) := x"59";
constant ccZ  : std_logic_vector(7 downto 0) := x"5A";
constant ccNL : std_logic_vector(7 downto 0) := x"0A";
constant ccCR : std_logic_vector(7 downto 0) := x"0D";


type ctrl_states is (start,state1,busy,busy1,stop);
signal  present_state   : ctrl_states ;


signal char_count : integer range 0 to 32;


begin

ascii_tx_string(0)  <= ccR
ascii_tx_string(1)  <= ccE ;
ascii_tx_string(2)  <= ccA ;
ascii_tx_string(3)  <= ccD ;
ascii_tx_string(4)  <= ccY ;


p1:process(CLK)
begin

if(CLK'event and CLK = '1') then


   if (RST = '0' ) then
       LCD_INIT <= '0';
       LCD_CMD  <= '0';
       LCD_DAT  <= '0';
       LCD_ON   <= '1';
       present_state <= start;
       char_count <= 0;

    else

    case present_state is


        when busy1 =>
            present_state <= state1;


        when busy =>
            if(LCD_IF = '1')then
            present_state <= state1;
```

```
                    LCD_INIT <= '0';
                    LCD_CMD  <= '0';
                    LCD_DAT  <= '0';
            end if;


        when state1 =>

        if(char_count <= 4) then
           LCD_CD <= ascii_tx_string(char_count);
           char_count  <= char_count + 1;
           LCD_INIT <= '0';
           LCD_DAT <= '1';
           present_state <= busy;

        end if;


        when start =>
                 char_count <= 0;
                 LCD_INIT <= '1';
         present_state <= busy;

        when others => null;

    end case;
end if;
end if;
end process;
end lcd_Controller_A;
```

Curriculum-Vitae

Name	: Bhagyasree M R
Address	: 75-D, Pocket A/3, Mayur Vihar Phase 3, New Delhi, Pin-110096
Contact Phone	: 9567329486
e-mail id	:bhagyasreemr92@gmail.com