P-2568

DESIGN AND EVOLUTION OF GENERALIZED FAULT DIAGNOSIS SYSTEM FOR DISTRIBUTED EMBEDDED SYSTEM

By

P. Suma

Reg.No:0720108021

Of

KUMARAGURU COLLEGE OF TECHNOLOGY

COIMBATORE-641006

(An Autonomous Institution affiliated to Anna University Coimbatore)

A PROJECT REPORT

Submitted to the

FACULTY OF INFORMATION AND COMMUNICATION ENGINEERING

In partial fulfillment of the requirements

For the award of the degree

of

MASTER OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

MAY 2009

BONAFIDE CERTIFICATE

Certified that this project report entitled "DESIGN AND EVOLUTION OF GENERALIZED FAULT DIAGNOSIS SYSTEM FOR DISTRIBUTED EMBEDDED SYSTEM" is the bonafide work of Ms. P. SUMA, who carried out the research under my supervision. Certified further, that to best of my knowledge the work reported here in does not from any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

Signature of the Guide

Dr.S.Thangasamy, Ph.D.,

Department of Computer

Science and Engineering

Head of the Department

Dr.S.Thangasamy, Ph.D.,

Department of Computer

Science and Engineering

The candidate with **University Register No. 0720108021** was examined by us in the project viva-voce examination held on 5/5/2009

5/5/09

INTERNAL EXAMINER

5/5/09

EXTERNAL EXAMINER

Government of India, Ministry of Defence
Defence Research & Development Organization
NAVAL PHYSICAL & OCEANOGRAPHIC
LABORATORY
Thrikkakara P.O, Cochin, India
Pin– 682 021

CERTIFICATE

This is to certify that the dissertation entitled **"Design and Evolution of Generalized Fault Diagnosis System for Embedded Systems"** is an authentic record of the work carried out at Naval Physical & Oceanographic Laboratory, DRDO, Ministry of Defence, Cochin-21, by **Ms. P. Suma** under my guidance and supervision in partial fulfilment of the requirements for the award of the degree of "Master of Engineering in Computer Science and Engineering ", Kumaraguru College of Technology, Coimbatore.

Dated 28 April 2009.

C.P. Kuriakose, Scientist-E
Project Guide

Countersigned

Lasitha Ranjit, Scientist-E
HRD Coordinator

The Institution of Electronics and Telecommunication Engineers

KOCHI CENTRE

XL/216K, 3rd Floor, Jewel Arcade, Layam Road, Kochi-682 011
Telefax: (+)91-484-2369944 E-mail: ietekochi@dataone.in

वीर्यं करवावहै

14 February 2009

Certificate of Presentation

This is to certifying that Ms. Suma. P, M. E. Computer Science and Engineering, Kumaraguru College of Technology, Coimbatotre has presented the Paper entitled ***Evolution and Design of Generalized Fault Diagnosis System in a Distributed Embedded System*** *in the Seminar organized by IETE Kochi Centre on 14 February 2009.*

14/2/09

Hon. Secretary
IETE Kochi Centre

Chairman
IETE Kochi Centre

IETE KOCHI CENTRE

DECLARATION

"I hereby declare that this thesis is my own work and effort and that it has not been submitted anywhere for any award. Where other sources of information have been used, they have been acknowledged."

Place: Coimbatore

Date:

Suma

Signature

ABSTRACT

The objective of this project is to design and evolve a framework of a fault diagnostic system for distributed embedded system. A case study of SONAR system is attempted. The system specification of an existing system is provided using a Graphical User Interface and thereby generates a fault tree. An illustrated graphical system layout will be generated for the existing system. Also a tree structure is generated to diagrammatically show the faulty node and also the path of dependent components causing the main system to fail. This helps in information management and thereby initiates fast remedial actions.

ஆய்வுச்சுருக்கம்

இந்த ஆய்வின் நோக்கமானது பரவலான உள்புகுத்தப்பட்ட டிஸ்ற்றிபுடெட் எம்பெட்டெட் அமைப்பின் பிழை கண்டுணரக்கூடிய ஒரு அமைப்பினை வடிவமைக்க உதவுகின்றது. ஏற்கனவே உள்ள அமைப்பின் குறியீடுகள் வரைகலை பயன்படுத்துனர் தொடர்பின் மூலம் பெறப்படுகிறது. மேலும் ஃபால்ட் ட்ரீ அமைப்பு ஒன்றினையும் உருவாக்குகின்றது. ஒரு விரிவுரைக்கப்பட்ட வரைகலை அமைப்பின் வடிவமைப்பானது ஏற்கனவே உள்ள அமைப்பின் மூலம் உருவாக்கப்படுகின்றது. மேலும் மரம் வடிவிலான அமைப்பு பிழையான முனைகள் மற்றும் பிழைகளை உருவாக்கக்கூடிய பாகங்களைக் கண்டுனர உதவுகின்றது. இது தகவல்கள் மேலாண்மை மற்றும் விரைவான பிழைதிருத்தல் பணிகளையும் ஊக்குவிக்கின்றது.

TABLE OF CONTENTS

INTRODUCTION

1. Introduction

An **embedded system** is a special-purpose computer system designed to perform one or a few dedicated functions, often with real-time computing constraints. It is usually *embedded* as part of a complete device including hardware and mechanical parts. In contrast, a general-purpose computer, such as a personal computer, can do many different tasks depending on programming. Embedded systems control many of the common devices in use today.

Embedded systems refer to devices, instruments or large engineering structures/systems that are built to handle one or a few pre-established tasks. The computer controlling the whole thing is built into or 'embedded' within the device.

Every embedded system consists of a Fault Diagnosis System (FDS) which consists of a critical node and other peer nodes. The critical node consists of the FDS for the entire system. The FDS polls the peer systems and collect information about their health. These data collected is then processed and faulty component is identified.

All other components dependent on the faulty component are identified and the fault path is traced. The knowledge about this faulty path is required by the operator to do maintenance work as soon as possible.

LITERATURE SURVEY

2.1 SONAR

2.1.1 Introduction

Sound is the propagation of mechanical energy in a material medium. It and can carry information. *Acoustic signal processing* is the discipline concerned with the separation of desired acoustic information from undesired information or noise. Acoustic sensing provides an excellent means for data gathering in the marine environment.

Acoustic signal processing can be two dimensional in the sense that it involves both *spatial* and *temporal* aspects. The temporal aspect concerns itself with the behavior in time of the signal or its spectral characteristics. The spatial aspect concerns itself mainly with the directional properties of the signal. This is accomplished by means of transducers. Transmitting transducers are called *projectors* and receiving transducers are called *hydrophones*. A collection of transducers is called an *array*.

SONAR (SOund, NAvigation, and Ranging) is a detection and location method based on the propagation of acoustic waves between the target and detector. It was developed as a means of tracking enemy submarines during World War II.

2.1.1.1 Types of SONAR

There are two types of SONAR.

Active SONAR

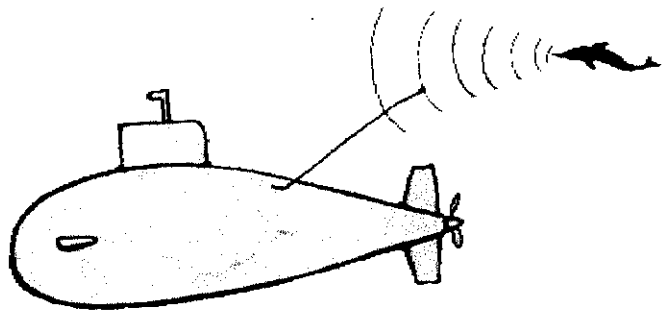
An electrical impulse from a transmitter is converted into a sound wave by the transducer and sent into the water. When this wave strikes an object, it rebounds. This echo strikes the transducer, which converts it back into an electric signal, which is amplified and processed by the receiver and sent to the display. There are two types of active systems depending on the location of source or receiver.

1) Monostatic

2) Bistatic

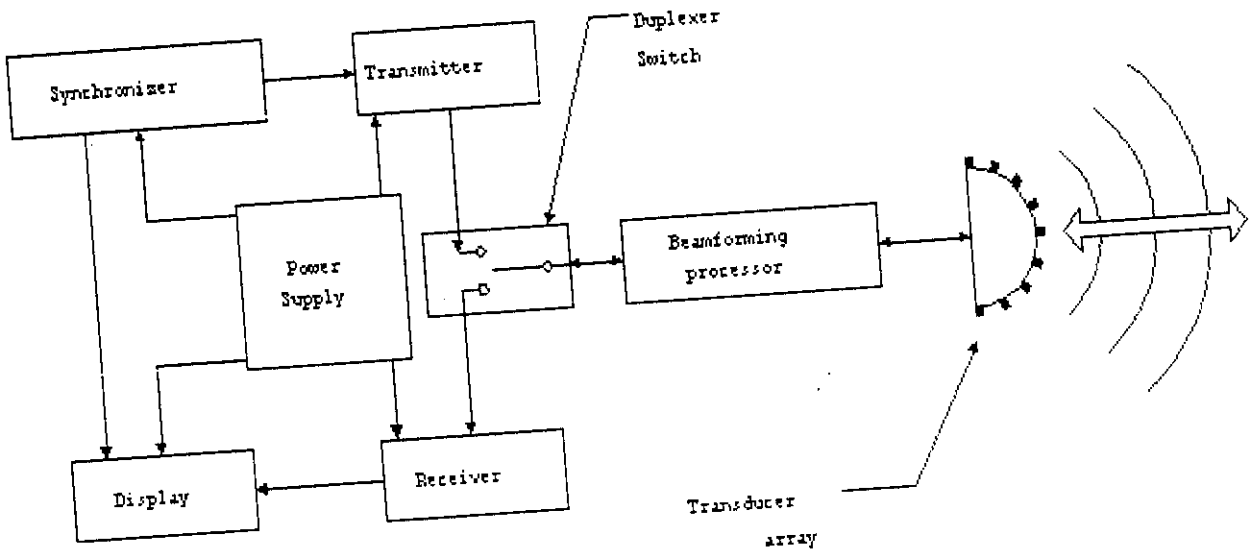
Monostatic sonar system has source/receiver located on same platform and both may employ same acoustic transducers. *Bistatic* system has separate source/receiver.

Since the speed of sound in water is constant (approximately 4800 feet per second), the time lapse between the transmitted signal and the received echo can be measured and the distance to the object determined. This process repeats itself many times per second.



Active Sonar

A functional diagram of an active sonar system looks like this:



The functional components are described below:

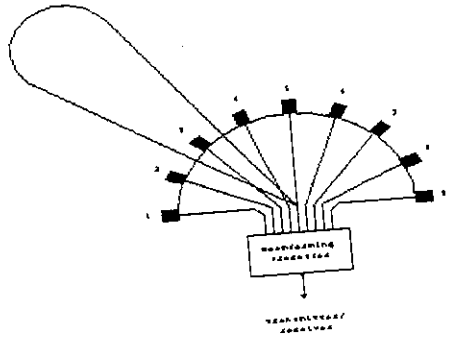
Beamforming processor: Pulses generated from a wave generator is modulated to an operating frequency and then beamformed by an array of transducers that focuses the signal in specific direction.

Transmitter: The signal is amplified and converted from an electric signal to sound wave by the transmitter transducers.

Receiver: The SONAR receiver implements operations similar to the transmitter. Hydrophones convert the acoustic signal to an electric one whereupon it undergoes some 'signal conditioning' to amplify it to an appropriate level. The receiver beamformer focuses energy arriving from specific directions for spatial imaging. The signal is further demodulated to low frequency.

Both the transmitter and receiver beamformers provides spatial resolution for the sonar system.

Transducer array: The individual transducers are simple elements with little or no directionality. By combining transducers in an array, one controls the directional properties of acoustic transmission and reception.



Active beamforming

Duplexer: The duplexer protects the receiver from the full transmitter power while the

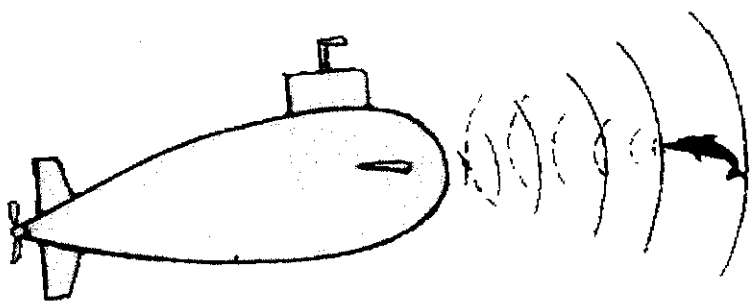
pulse is going out. It can be thought of a switch that toggles between the transmitter and receiver.

Display: Based on the display algorithm used, the low frequency signal is further normalized and displayed.

Synchronizer: Provides overall coordination and timing for the system. Reset the display for each new pulse in order to make range measurements.

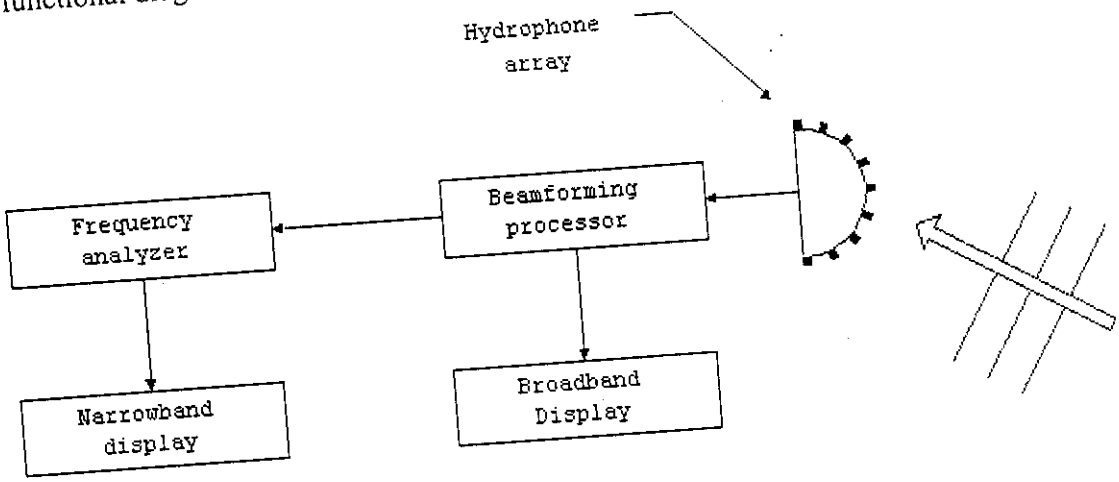
Passive SONAR

A passive sonar system, only listens and do not transmit. Passive sonar does not produce its own acoustics. Here, the source itself is the target. A target can be detected by the noise it makes, from its machinery, the propeller, or the sound of the water passing around the vessel as it travels.



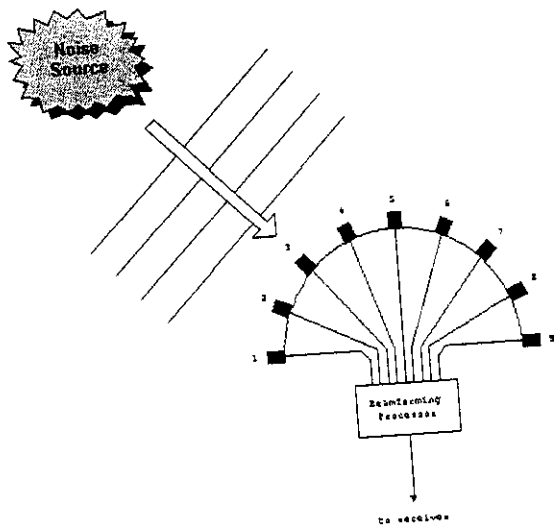
Passive Sonar

A functional diagram of an active sonar system looks like this:

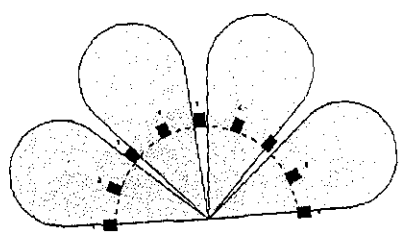


Hydrophone array: These are the sensitive elements which detect the acoustic energy emitted from the target. Again, they are arranged into an array to improve the beam width. Common configurations are cylindrical or spherical.

Beamforming processor: Unlike active systems which transmit and receive in a set direction, the passive system must listen to all angles at all times. This requires a very wide beam width. At the same time, a narrow beam width is required for locating the source and rejecting ambient noise. These two objectives are achieved simultaneously by the passive beamforming processor.



Passive hydrophones



Passive beamforming

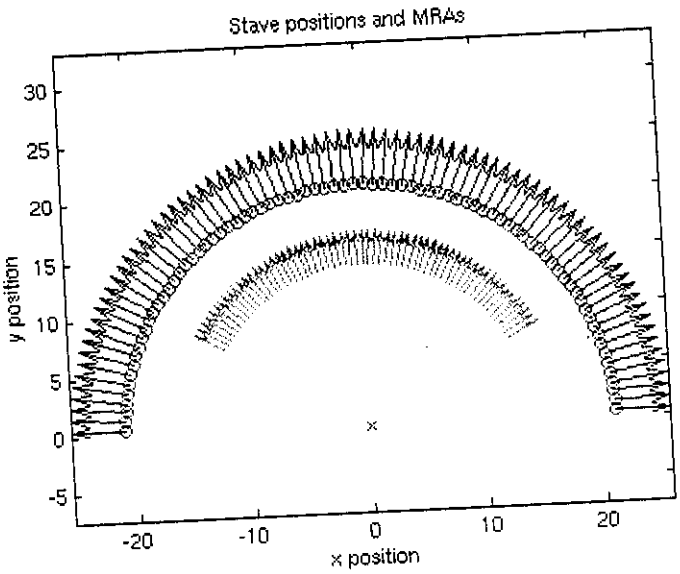
Broadband display: The output of the beamforming processor is displayed as a bearing time history.

Frequency Analyzer: The frequency analyzer breaks the signal into separate frequencies. This is the spectrum of the signal. For processing purposes, the frequencies are divided into small bands known as *frequency bins*. The width of each bin is called the analysis bandwidth. The frequency analyzer separates (filters) the signal into discrete bins.

2.1.1.2 Beamforming

Beamforming is a signal processing technique used in sensor arrays for directional signal transmission or reception. This spatial selectivity is achieved by using adaptive or fixed receive/transmit beam pattern. Beamforming is the process of trying to concentrate the array to sounds coming from only one particular direction. It is a spatial filter that operates on the output of an array of sensors in order to enhance the amplitude of a coherent wavefront relative to background noise and directional interference.

The figure below shows a curved array of hydrophone sensors, or staves. Each sensor (red circle) is located at an (x,y) coordinate as shown. These sensors are pointed in known directions (blue arrows), and we wish to form beams which point in chosen directions (green arrows). The "pointing direction" is called the Maximum Response Angle (MRA), and can be arbitrarily chosen for the beams.



The response of a given element is plotted on a polar graph, where the angle is the offset from the MRA, and the radius is the magnitude response (dB) in that direction.

The goal of beamforming is to sum multiple elements to achieve a narrower response in a desired direction (the MRA). That way when a sound is heard in a given beam, its direction is known.

Why sound energy?

In the dull, dense and conductive environment of seawater, electromagnetic waves and light rays are severely attenuated. Acoustic energy however has proven to be more tolerant and controllable in this medium, but the sound is still influenced in many ways during its passage through the sea, consequently substantial acoustic energy is essential for many sonar applications

2.2 Fault Tree Analysis (FTA)

2.2.1 Introduction

Fault tree analysis (FTA) is a graphical top-down approach to failure analysis, starting with a potential undesirable event (accident) called a TOP event, and then determining all the ways it can happen. The analysis proceeds by determining how the TOP event can be caused by individual or combined lower level failures or events. The causes of the TOP event are connected through logic gates. FTA is the most commonly used technique for causal analysis in risk and reliability studies.

2.2.2 FTA main steps

- Definition of the system, the TOP event (the potential accident), and the boundary conditions
- Construction of the fault tree
- Identification of the minimal cut sets
- Qualitative analysis of the fault tree
- Quantitative analysis of the fault tree
- Reporting of results

2.2.3 Preparation for FTA

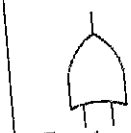
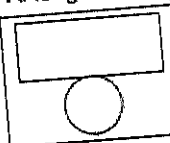
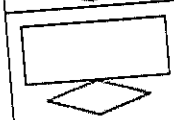
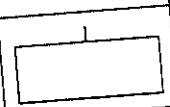
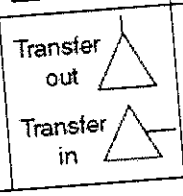
- The starting point of an FTA is often an existing Failure Modes, Effects and Criticality Analysis (FMECA) and a system block diagram.
- FMECA is an essential first step in understanding the system.
- The design, operation, and environment of the system must be evaluated.
- The cause and effect relationships leading to the TOP event must be identified and understood.

2.2.4 Fault Tree Building Blocks

In order to find and visualize casual relation by fault trees, we require building blocks to classify and connect a large number of events. There are two types of building blocks.

a) Gate symbols

b) Event symbols

Logic gates	 OR-gate	The OR-gate indicates that the output event occurs if any of the input events occur
	 AND-gate	The AND-gate indicates that the output event occurs only if all the input events occur at the same time
Input events (states)		The basic event represents a basic equipment failure that requires no further development of failure causes
		The undeveloped event represents an event that is not examined further because information is unavailable or because its consequences are insignificant
Description of state		The comment rectangle is for supplementary information
Transfer symbols	 Transfer out Transfer in	The transfer-out symbol indicates that the fault tree is developed further at the occurrence of the corresponding transfer-in symbol

2.2.5 Fault Tree construction

The most undesired event for the system or its mission is chosen as the topmost event. The fault tree logic diagram is constructed relating all possible sequences of events whose occurrence would produce the undesired event chosen. These casual factors are connected to the top event using the logic gates. The fault tree is continued till the last event which cannot be further analyzed called the basic event represented using a circle.

All the nodes and the top event is represented using a rectangle. Once the fault tree is constructed using the probability of failure of basic event, we can calculate the probability of occurrence of the top event.

2.2.6 Potential Application of FTA

Fault Tree Diagram is useful document for maintenance people in locating the probable cause of any particular type of failure. FTA leads to improved understanding of system characteristics. Design flaws and insufficient operational and maintenance procedures may be revealed and corrected during the fault tree construction. The quantitative analysis depicts the probability of any top event. The technique is particularly useful in safety analysis where the block diagramming discipline helps prevent oversights. The major advantage of FTA is its ability to address human errors, which the FMECA cannot address. FTA is not (fully) suitable for modeling dynamic scenarios. FTA is binary (failure/success) and may therefore fail to address some problems.

2.2.7 Qualitative Analysis of Fault Tree

Using the failure rates of functional modules(eg. PCBs), the probability of the occurrence of the top events is calculated. The probability of failure of a card is calculated from $\mathbf{p=1 - e^{-\lambda t}}$ where **p** is probability of failure, λ is the failure rate of the card and **t** is the mission time of the sonar.

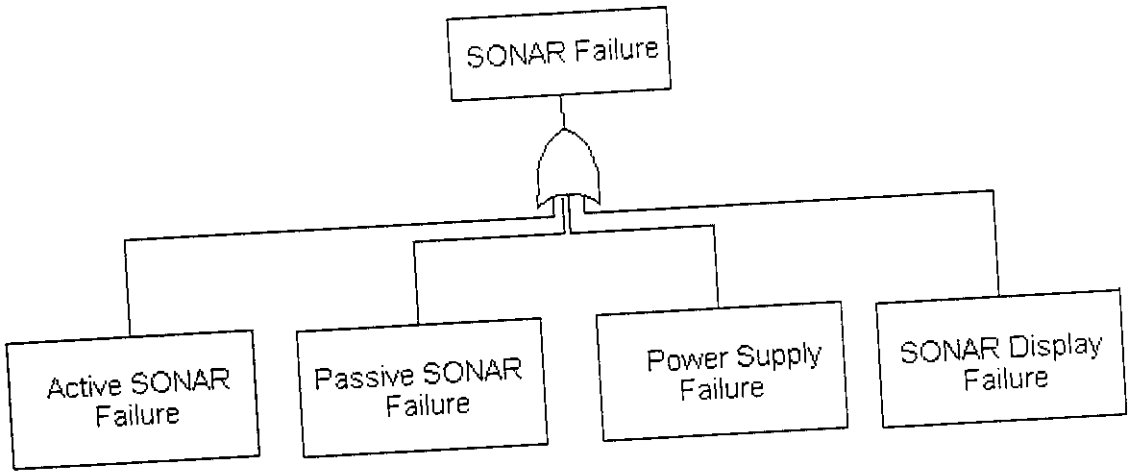
Failure probability calculation is carried out as follows:
The probability failure of an event connected by AND gates are multiplied to get the effective probability of that event. The probability failure of an event connected by OR gates are added together to get the effective probability of that event. Based on these principles, the probability of the failure of sonar is calculated. With this probability of failure, the reliability and availability of the sonar for a mission time is calculated.

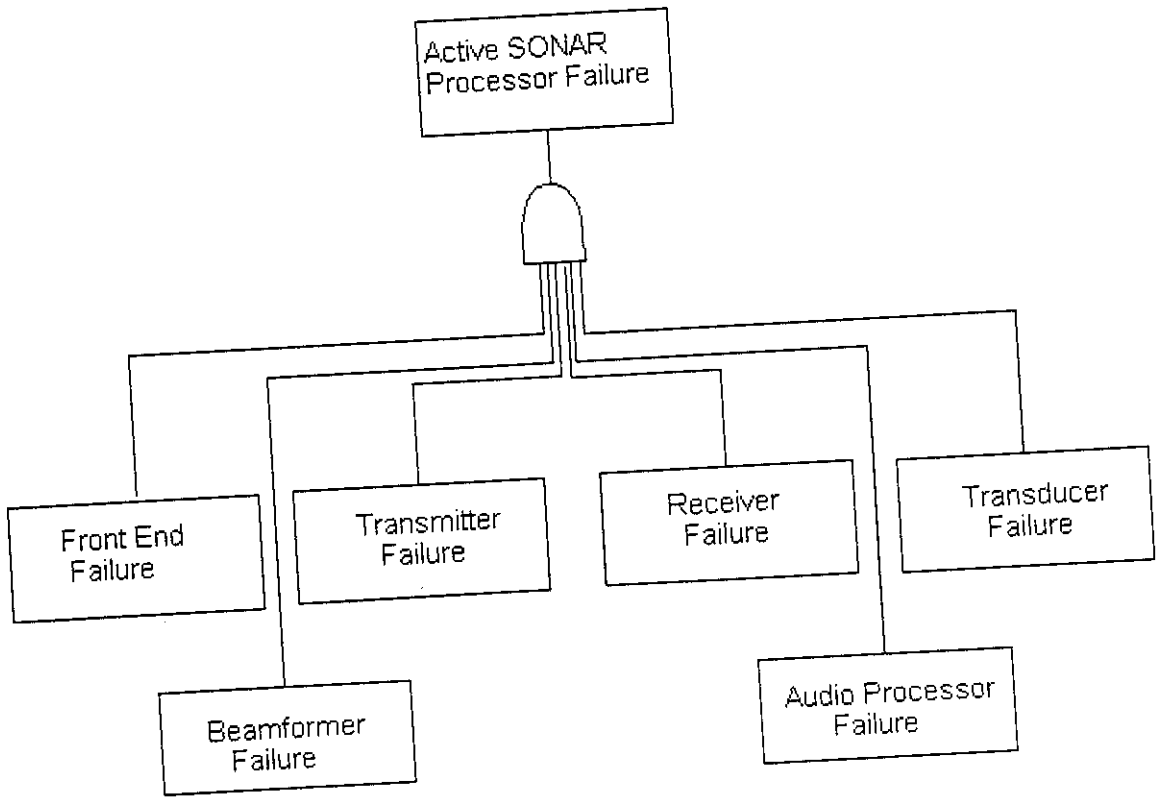
2.2.8 Example

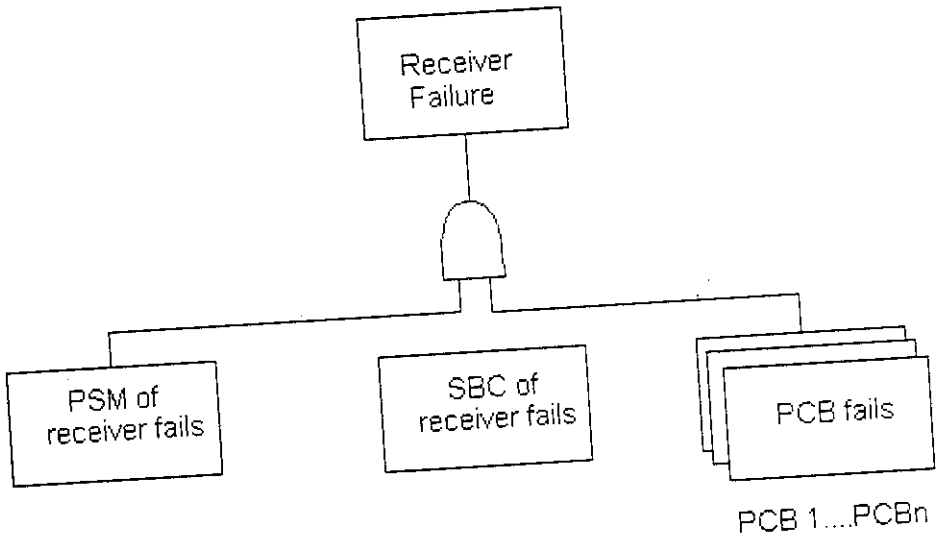
Top Event(TE)	Subsequent events(SE)
SONAR failure	Active Sonar Processor Failure
	Passive Sonar Processor Failure
	SONAR Display Failure
	Power supply Failure
Local Health Monitoring System Failure	

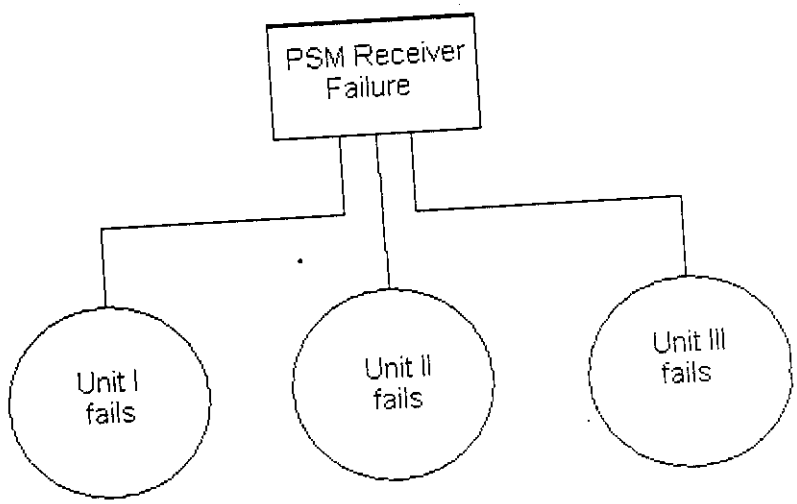
Top Event(TE) .	Subsequent events(SE)
Active Sonar Processor Failure	Front End Failure
	Beam former Failure
	Transmitter Failure
	Receiver Failure
	Audio Processor Failure
	Transducer Failure

Fault Trees


SONAR Failure
Active SONAR Failure
Passive SONAR Failure
Power Supply Failure
SONAR Display Failure


Active SONAR Processor Failure
Front End Failure
Transmitter Failure
Receiver Failure
Transducer Failure
Beamformer Failure
Audio Processor Failure

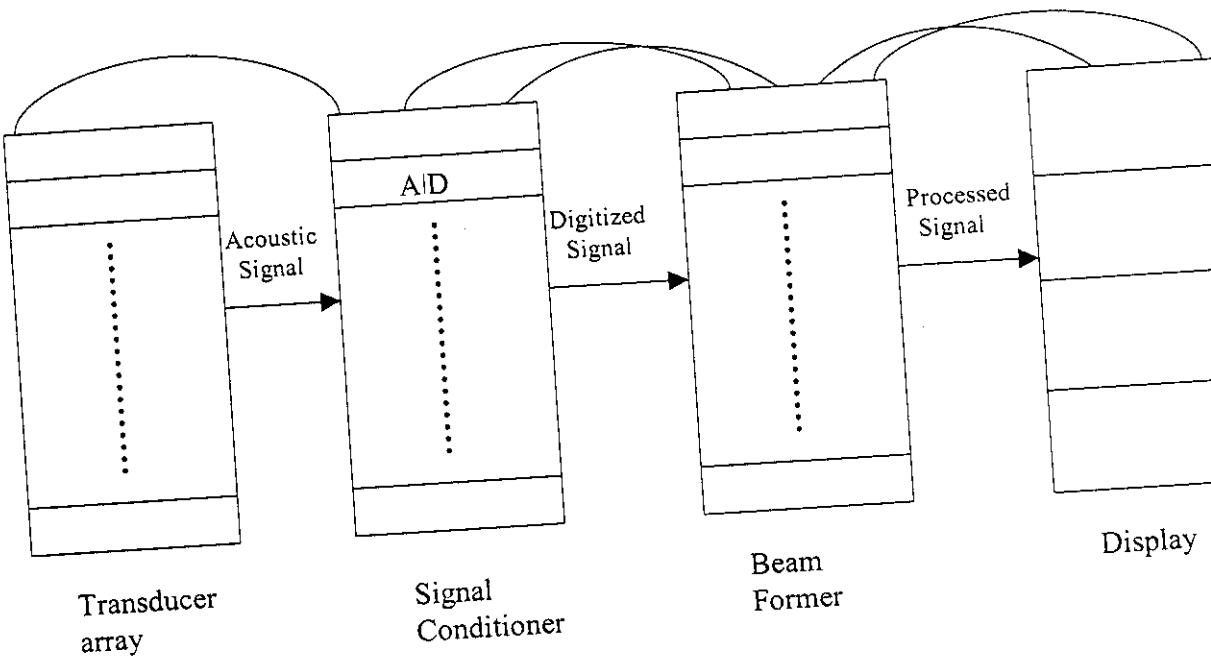

Receiver Failure
PSM of receiver fails
SBC of receiver fails
PCB fails
PCB 1....PCBn


PSM Receiver Failure
Unit I fails
Unit II fails
Unit III fails

2.3 System Overview

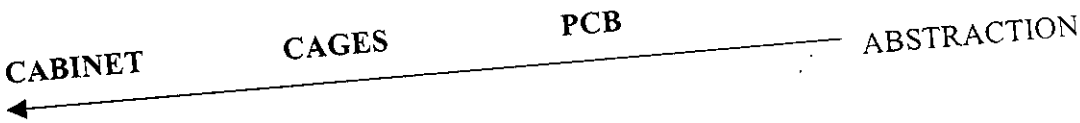
2.3.1 Introduction

The physical plant is a network of distributed embedded system interconnected using Ethernet. Each node of the network, performs various tasks and is realized using electronic cabinets and display units. Each of these cabinets encloses many PCBs performing various tasks (heterogeneous) which are arranged within different cages or levels in a cabinet.

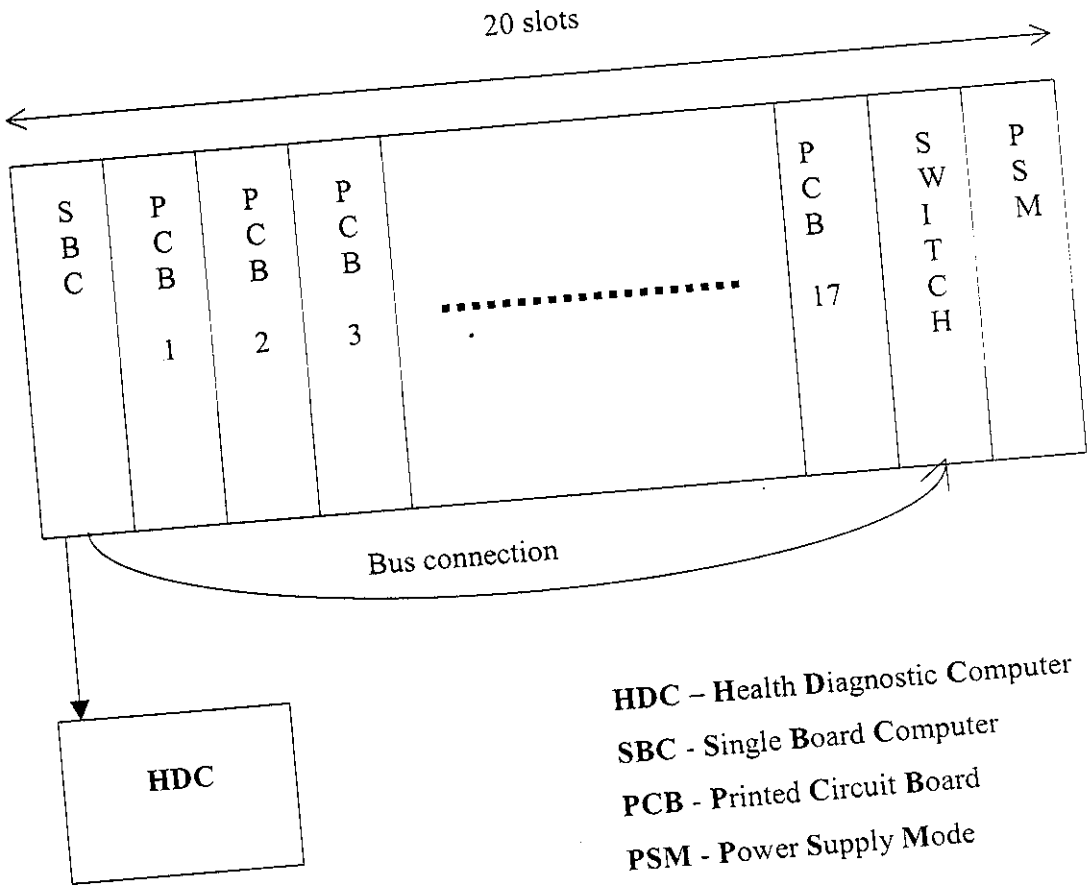


2.3.2 Hierarchy

In abstraction hierarchy, cabinet is the one with highest abstraction and it reduces with lower level of hierarchy from cages to PCBs



2.3.3 Cage Backplane



HDC – **H**ealth **D**iagnostic **C**omputer

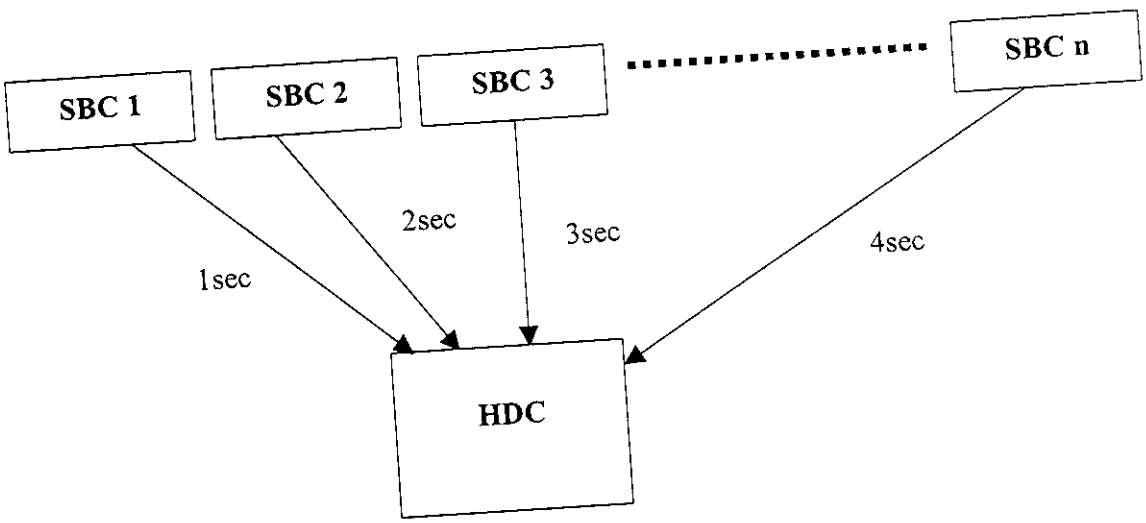
SBC - **S**ingle **B**oard **C**omputer

PCB - **P**rinted **C**ircuit **B**oard

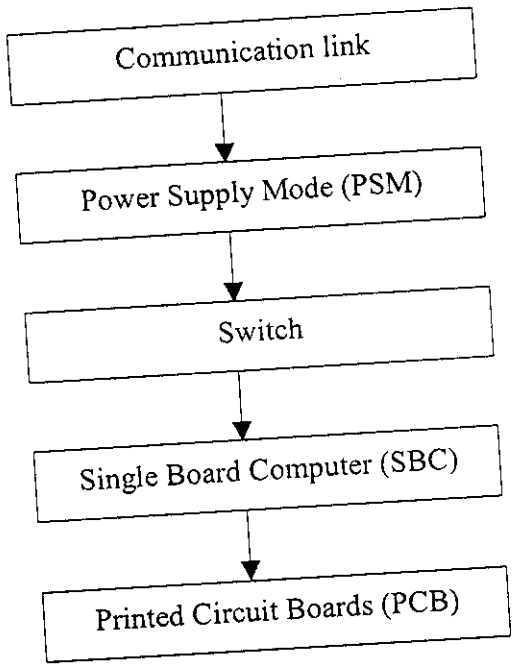
PSM - **P**ower **S**upply **M**ode

2.3.4 Health Diagnostic Computer (HDC) and Local Health Monitoring System (LHMS)

Every cabinet has its own Local Health Monitor Module (LHMM) which diagnoses and sends its local health status to the Health Diagnostic Computer (HDC) at some periodicity. The HDC may also poll each of the SBCs in a round-robin manner based on the time interval set from the Human Machine Interface (HMI).



2.3.4.1 Hierarchy of diagnosis



Cabinet level data links need to be connected to HDC – no further diagnosis required

PSM failure indicates failure of the cage – no further diagnosis required

Switch failure - Observations from SBC cannot be routed to HDC – no further diagnosis required

SBC failure indicates observations from different PCBs cannot be collected - no further diagnosis required

No data or wrong data available from the failed PCB – final information will not be accurate and hence points to failure of PCB software.

2.3.5 PCB

Inside the cage, each PCB may or may not have a ***Built-in Self Test (BIST)*** based on its requirement and capability. BIST locally isolates a fault in the PCB and reports to the SBC which in turn goes to HDC. In the absence a BIST, fault isolation of the PCB may be done by software analysis where the inferences from the neighboring PCBs are used.

PROPOSED SYSTEM

3. Proposed System

The proposed system is to evolove a method of generalized fault diagnosis system for embedded systems, as an enhancement to now existing customized fault diagnosis system.

A GUI is to be designed to provide the complete system specification and details in a user interactive session. This information is used to generate an illustrated system layout depicting details of each component. I must also generate a tree structure of the subsystem dependencies and must depict the path of fault, causing system failure. Complete system details are recorded into a specified file. The proposed GUI helps modification/enhancements in hardware/software of existing systems and thus saves a lot of development time. System layout and fault tree analysis for the system is also generated in this process.

The GUI provides a method to enter information about various embedded system hardware and dynamically generate system layout.

SYSTEM DESIGN

4.1 Requirement Analysis and Design

4.1.1 Introduction

Real-time systems interact with their external environment. The set of external objects of significance and their interactions with the system form the basis of the requirement analysis of the system. In the Unified Modeling Language, this is captured in the form of use case model. A use case is a system capability that is detailed with accompanying text, examples (scenarios), or state models. The use case model decomposes the primary functionality of the system and the protocols necessary to meet these functional requirements.

4.1.2 Unified modeling Language (UML)

The Unified Modeling Language (UML) is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems. The UML is a very important part of developing objects oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects. Using the UML helps project teams communicate, explore potential designs, and validate the architectural design of the software.

Goals of UML

The primary goals in the design of the UML were:

1. Provide users with a ready-to-use, expressive visual modeling language so they can develop and exchange meaningful models.
2. Provide extensibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development processes.
4. Provide a formal basis for understanding the modeling language.
5. Encourage the growth of the OO tools market.

6. Support higher-level development concepts such as collaborations, frameworks, patterns and components.
7. Integrate best practices.

Why use UML?

As the strategic value of software increases for many companies, the industry looks for techniques to automate the production of software and to improve quality and reduce cost and time-to-market. These techniques include component technology, visual programming, patterns and frameworks. Businesses also seek techniques to manage the complexity of systems as they increase in scope and scale. In particular, they recognize the need to solve recurring architectural problems, such as physical distribution, concurrency, replication, security, load balancing and fault tolerance. Additionally, the development for the World Wide Web, while making some things simpler, has exacerbated these architectural problems. The Unified Modeling Language (UML) was designed to respond to these needs.

UML Diagrams

Each UML diagram is designed to let developers and customers view a software system from a different perspective and in varying degrees of abstraction. UML diagrams commonly created in visual modeling tools include:

Use Case Diagram displays the relationship among actors and use cases.

Class Diagram models class structure and contents using design elements such as classes, packages and objects. It also displays relationships such as containment, inheritance, associations and others. [1]

Interaction Diagrams

- **Sequence Diagram** displays the time sequence of the objects participating in the interaction. This consists of the vertical dimension (time) and horizonta

- **Collaboration Diagram** displays an interaction organized around the objects and their links to one another. Numbers are used to show the sequence of messages.

State Diagram displays the sequences of states that an object of an interaction goes through during its life in response to received stimuli, together with its responses and actions.

Activity Diagram displays a special state diagram where most of the states are action states and most of the transitions are triggered by completion of the actions in the source states. This diagram focuses on flows driven by internal processing.

Physical Diagrams

- **Component Diagram** displays the high level packaged structure of the code itself. Dependencies among components are shown, including source code components, binary code components, and executable components. Some components exist at compile time, at link time, at run times well as at more than one time.
- **Deployment Diagram** displays the configuration of run-time processing elements and the software components, processes, and objects that live on them. Software component instances represent run-time manifestations of code units.

4.1.2 Requirement Specification

1. Design a GUI to provide the complete system details.

2. Generate an illustrated graphical system layout from the system information made available through the GUI.

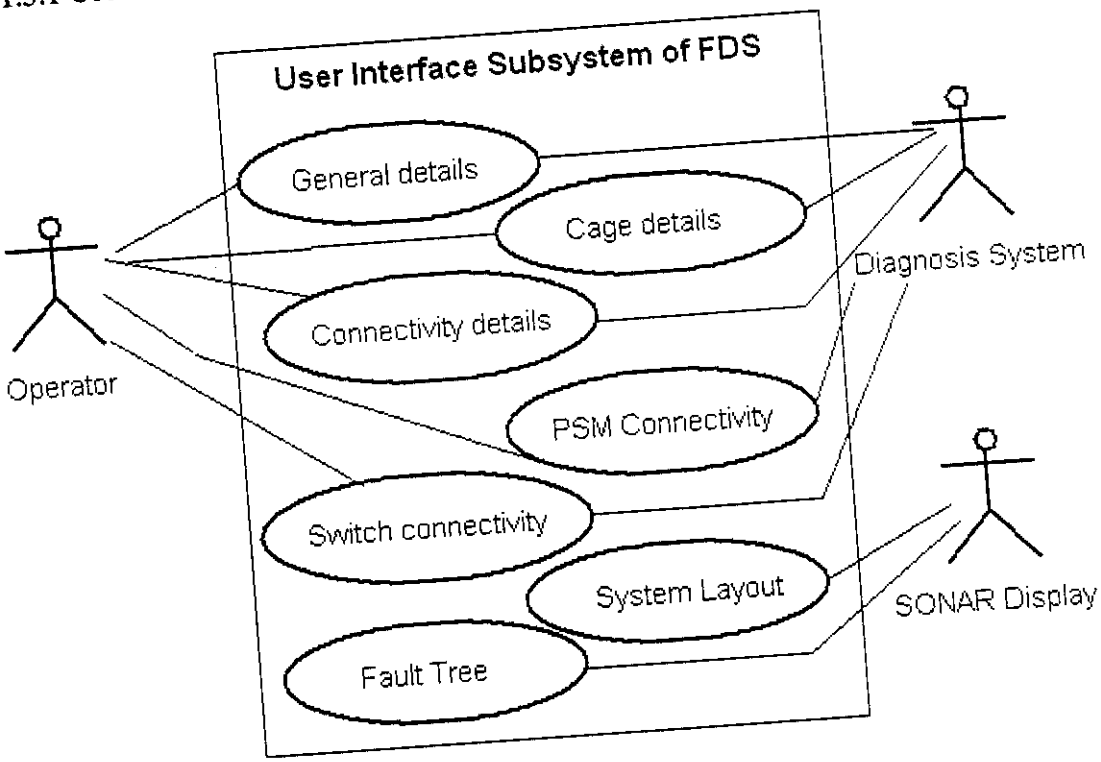
3. Differentiate system components with various color codes and illustrate their status.

4. Generate a tree structure for the subsystem dependencies and depict occurrences of faults, thus forming a fault tree.

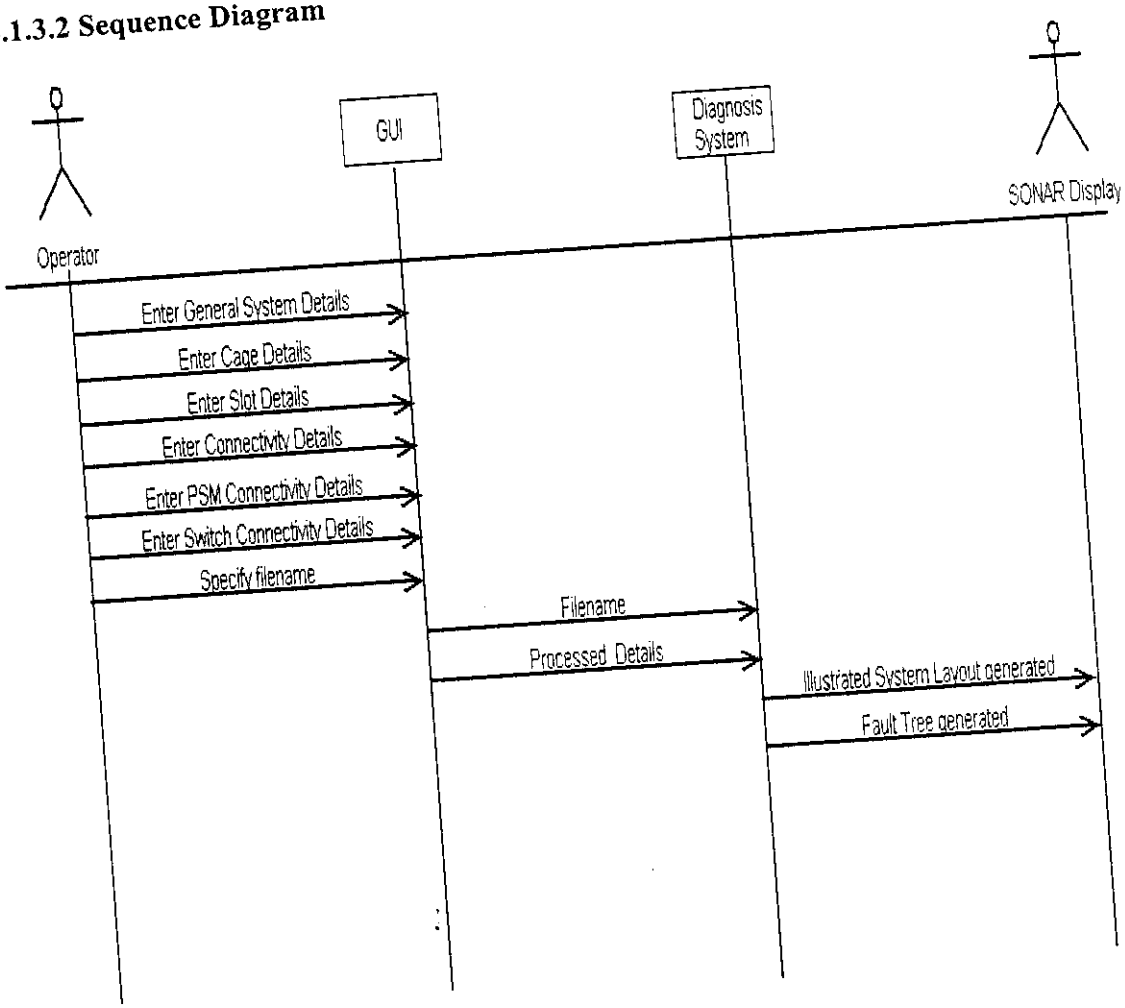
5. The system component status and other details and specifications are to be recorded in a file specified.

4.1.3 Diagrams

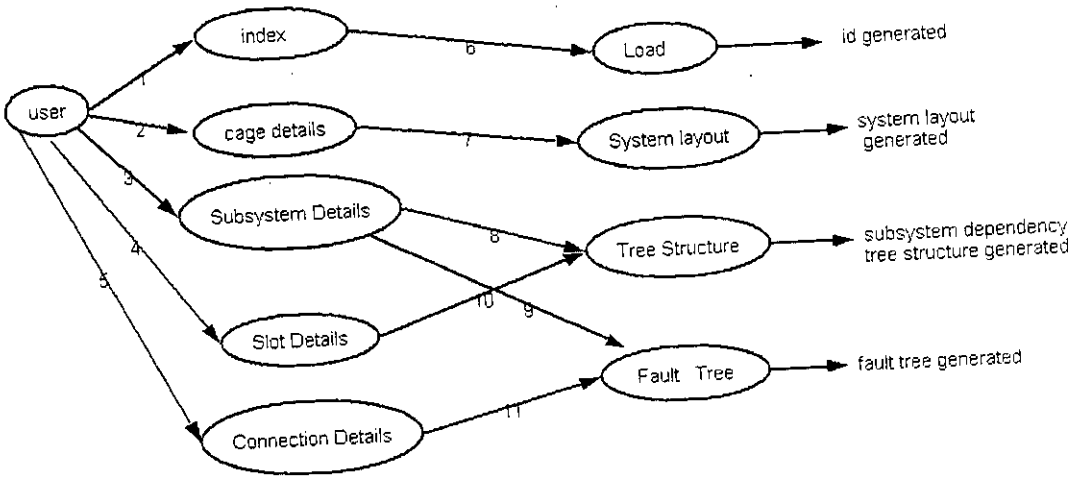
4.1.3.1 Use case Diagrams



4.1.3.2 Sequence Diagram



Data Flow Diagram(DFD)



1. no. of cabinets,slots,psm,switch,subsystems
2. no. of slots
3. subsystem name
4. function of module
5. inter slot connection details
6. ids
7. slot details
8. subsystem name
9. subsystem name
10. subsystem dependency
11. faults

CONCLUSION

5.1 Result and Discussion

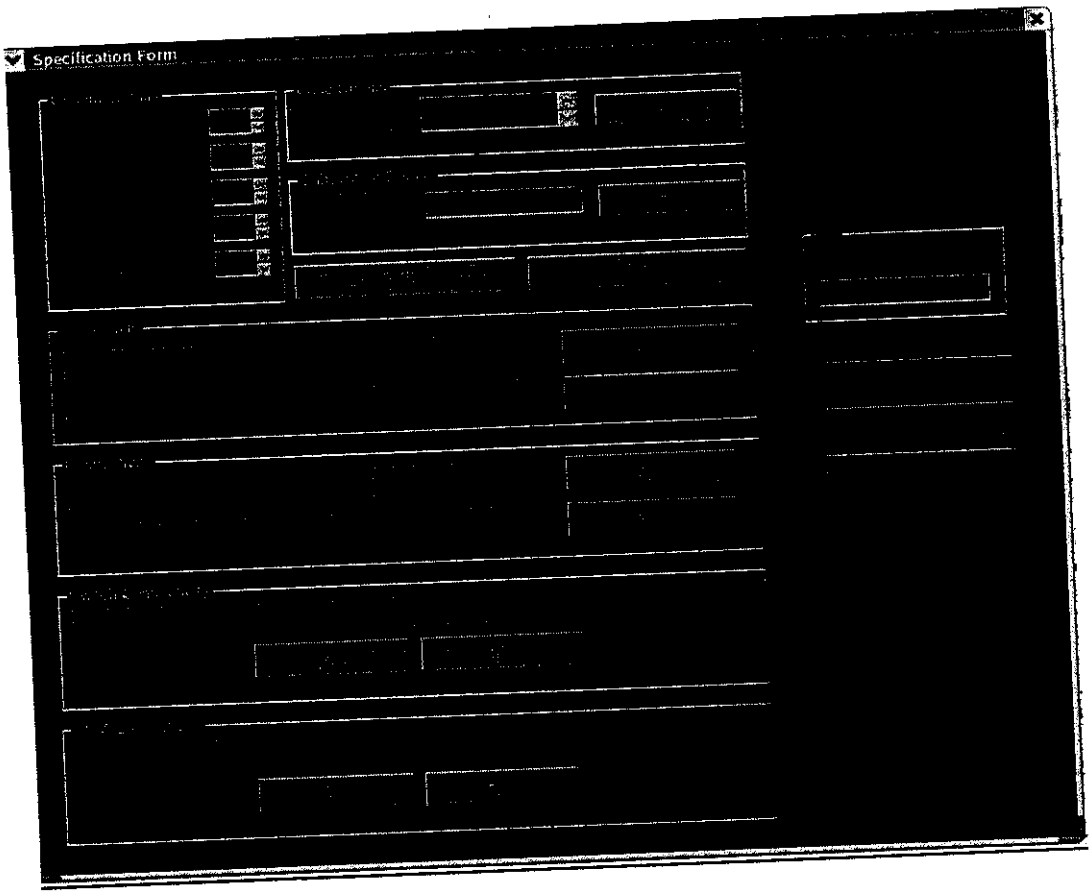
The GUI for providing details of the system is designed and implemented for the purpose of generalization of the system.

The illustrated system layout is generated from the information gathered through the GUI. Illustration is done using various color codes to differentiate different components of the system.

The tree structure is designed and implemented to depict the faulty component and other dependent components at fault causing the system to fail.

Complete system details are recorded into the specified file.

SCREENSHOTS



Enter general details like:

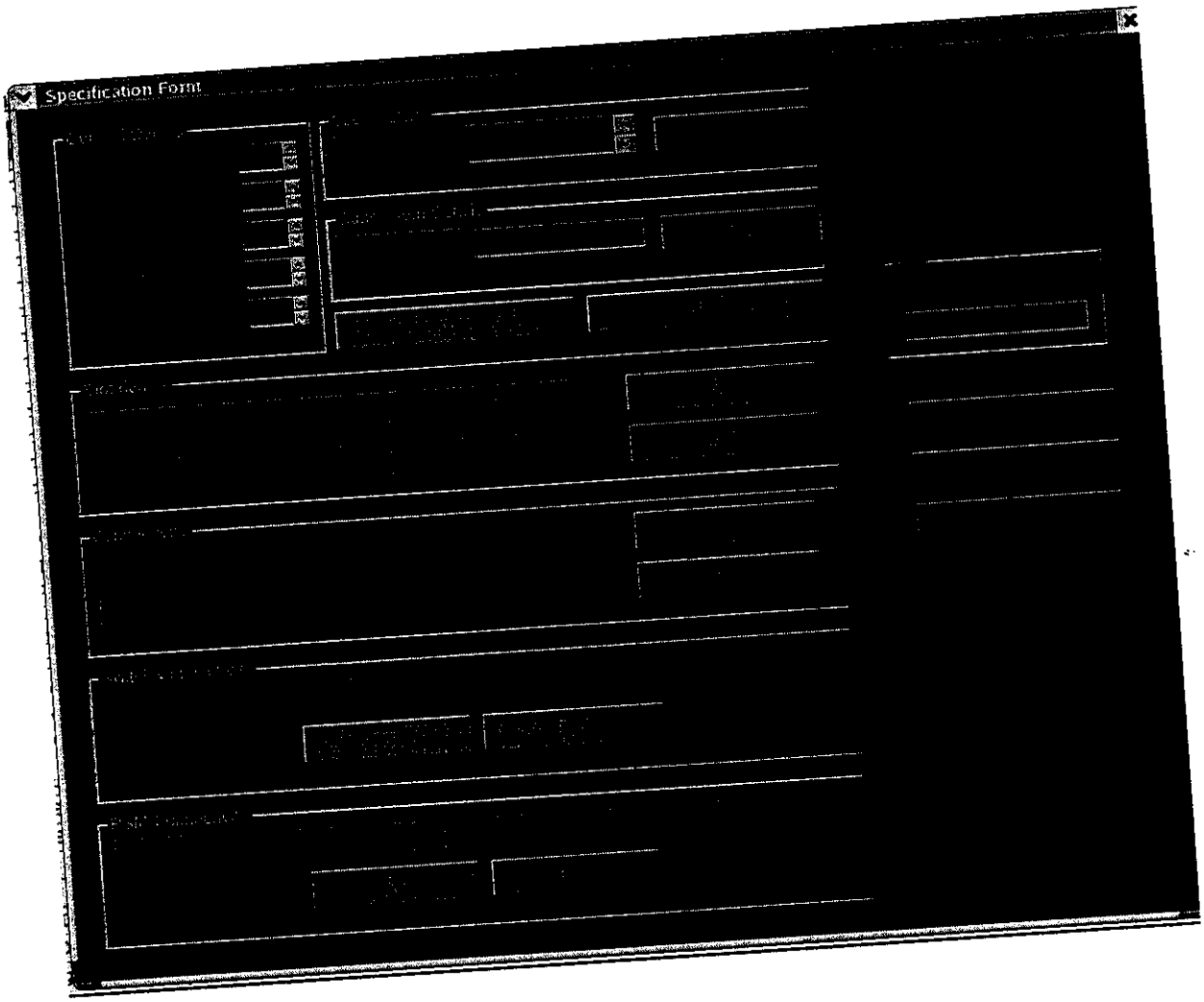
No. of switches

No. of PSM

No. of Cabinets

No. of Subsystem

No. of slots



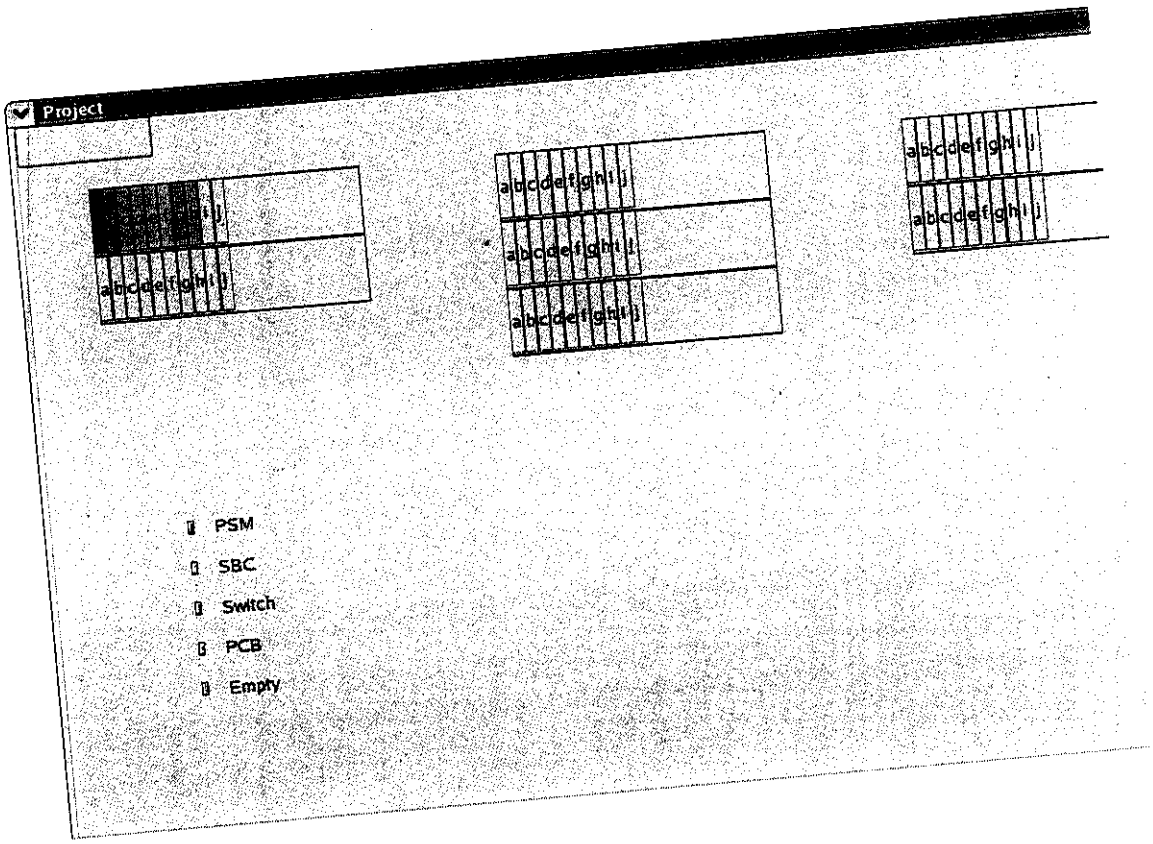
Enter number of cages in each cabinet and also provide subsystem details line subsystem name.

Generate index and Load details.

Provide slot functional details and inter slot connectivity details.

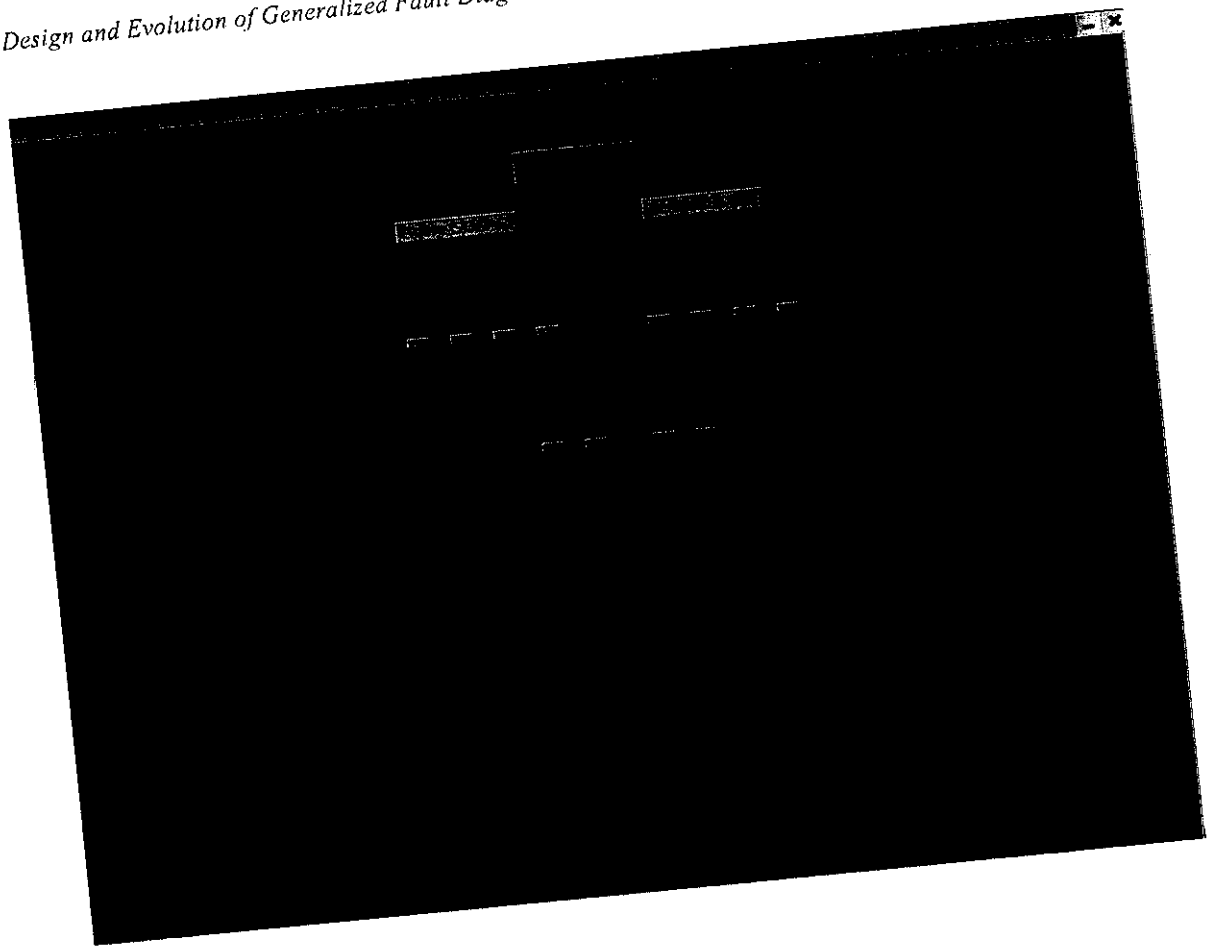
Also provide switch and PSM connectivity details.

Specify the filename wherein the details are to be recorded.



System layout is generated.

Illustration of slot status provided using tool tips.



Tree structure of subsystem dependency is generated.

Fault at PSM module at the lowest level causes all the depended slots in the next higher to fail causing the corresponding subsystem and ultimately the system to fail.

5.2 Sample Code

Datastructures used

```
int n,nswitch,nPSM,ncabinet,nslot,nsubsys=4;

//PSM
struct PSM
{
  QString PSM_id;
  QString component[25];
}P[10];

//Switch
struct Switch
{
  QString switch_id;
  QString component[25];
}SW[10];

//Cabinet
struct Cabinet
{
    QString cab_id;
    int ncage;
    int cabx;
    //Cage
    struct Cage
    {
        QString cage_id;
      int cagey;
          //Slot
     struct Slot
          {
              QString slot_id;
          }S[100];
    }Cg[20];
  }Cb[20];

  //Subsystem

  struct Subsystem
  {
      QString ssid;
      QString ssname;
      QString component[25];
   }SS[10];

   //Record
   struct Record
   {
       QString cabinet;
       QString cage;
       QString slot;
```

```
    QString specifictype;
}R[100];

//Connection Record
struct ConnectionRecord
{
    QString src;
    QString dst;
    QString status;
}Conn_Rec[125];

//Switch Record
struct SwitchRecord
{
    QString c_switch;
    QString slot;
    QString status;
}SW_Rec[125];

//PSM Record
struct PSMRecord
{
    QString c_psm;
    QString slot;
    QString status;
}PSM_Rec[125];

// struct PSMRecord *psm;

char type[6][10]={"PSM","SBC","Switch","PCB","Empty"};
```

Code for generation of System Layout

```
void SpecForm::drawSL()
{
    QLabel *label=new QLabel(0);
    label->setFixedSize(1000,700);
    label->setFrameStyle( QFrame::Panel | QFrame::Raised );
    label->setPaletteBackgroundColor("black") ;
    label->setLineWidth( 2 );

    QLabel *lbl =new QLabel[20](label);

    int lblw=200,lblh=50; //label/cage wdth nd ht

    QPushButton *b =new QPushButton[300](label);

    int bw=10,bh=48; //button/slot wdth nd ht

    int x=50,y=50;
```

```
    char
ind[20]={'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p
','q','r','s','t'};

          tx=x;
          ty=y;

   int r=0;  //for incremental object creation for each iteration of
for loop for drawing cages
   int d=0;

 QString filename;
   filename=FilenamelineEdit->text();

  QFile file(filename);
  file.open(IO_ReadWrite | IO_Append);

  QDataStream out(&file);
  out.setVersion(5);

  for(int k=0;k<ncabinet;k++)          //drawing cabinets
     {
        for(int i=r;i<Cb[k].ncage+r;i++)          // drawing cages
      {
          lbl[i].move(tx,ty);
          lbl[i].setFrameStyle( QFrame::Box | QFrame::Plain );
          lbl[i].setPaletteBackgroundColor("light pink") ;
          lbl[i].setLineWidth( 1 );
          lbl[i].resize(lblw,lblh);
          ty=ty+lblh;
       }//close of drawing cages

        tx=tx+lblw+100;
        ty=50;
        r=r+Cb[k].ncage;
    } //close of drawing cabinets

   //cabx
    Cb[0].cabx=50;

    for(int i=1;i<ncabinet;i++)
    {
        Cb[i].cabx=Cb[0].cabx+lblw*i+100*i;
    }

    //cagey
    for(int i=0;i<ncabinet;i++)
        for(int j=0;j<Cb[i].ncage;j++)
          {
             if(i==0&&j==0)
            Cb[i].Cg[j].cagey=50;
             else
            Cb[i].Cg[j].cagey=Cb[0].Cg[0].cagey+lblh*j;
          }
```

```
for( k=0 ; k<ncabinet ; k++)
   {
     for( t=0 ; t<Cb[k].ncage ; t++)
       {
      tx=Cb[k].cabx;
      ty=Cb[k].Cg[t].cagey;
      for( j=d,p=0 ; j<nslot+d; j++,p++)     // drawing slots
                 {
               b[j].setText("P\nC\nB\n\n");
                    b[j].setFlat(true);
               QString str;
               str=b[j].text();
               str.append(ind[p]);
               b[j].setText(str);
               b[j].move(tx,ty);
               b[j].resize(bw,bh);
               tx=tx+bw;
               QPushButton *btn=&b[j];
               QString strng=Cb[k].Cg[t].S[p].slot_id;
                       QString strng1=strng;
                 QString compstr=" ";
                 for(int g=0;g<nsubsys;g++) {
                     for(int h=0;SS[g].component[h]!='\0';h++)

      if(SS[g].component[h]==Cb[k].Cg[t].S[p].slot_id)
                              compstr.append(SS[g].ssname); }
                              strng.append(compstr);

  if((PSM_Rec[p].slot==strng1)&&(PSM_Rec[p].status=="Connected"))
                {
                    strng.append(" ");
                    strng.append(PSM_Rec[p].c_psm);
                 }

   if((SW_Rec[p].slot==strng1)&&(SW_Rec[p].status=="Connected"))
                 {
                     strng.append("  ");
                     strng.append(SW_Rec[p].c_switch);
                  }

                   QToolTip::setGloballyEnabled ( true );
                          QToolTip::add(btn,strng);
                     strng.append("\n");
                          btn->show();
                          if(!strng.isEmpty())
                          out<<strng;
                      } //close of drawing slots
                     d=d+nslot;

            }
          } //close of cab for loop
       label->show();
     } //close of drawpic


    void SpecForm::load_subsys()
```

```
    for(int i=0;i<nsubsys;i++)
        if(SSidcomboBox->currentItem()==i)
                        SS[i].ssname=SSnamelineEdit->text();

    SSnamelineEdit->clear();
}

void SpecForm::setSS()
{
    nsubsys=SubsysspinBox->value();
     char subsys_Index[6]={ '1' , '2' , '3' , '4' , '5' , '6' };

   //Subsystem Index
   QString ss="SS";
   for(int i=0;i<nsubsys;i++)
   {
       ss.append(subsys_Index[i]);
       SS[i].ssid=ss;
       ss="SS";
    }

     SSidcomboBox->clear();

    for(int i=0;i<nsubsys;i++)
         SSidcomboBox->insertItem(SS[i].ssid,-1);


   }


   void SpecForm::drawFT()
   {
    QLabel *labl1 =new QLabel(0);
    labl1->setFixedSize(1000,700);
    labl1->setFrameStyle( QFrame::Panel | QFrame::Raised );
    labl1->setPaletteBackgroundColor("black") ;
       int x=400,y=100;
       QPushButton *mainbt =new QPushButton(labl1);
       mainbt->resize(100,30);
       mainbt->move(500,50);//100
       mainbt->setText("System Fail");
       mainbt->setPaletteBackgroundColor("red") ;
       QPushButton *ssbt =new QPushButton[10](labl1);
       QPushButton *pcbbt =new QPushButton[200](labl1);
       QPushButton *psmbt =new QPushButton[20](labl1);
       QPushButton *swbt =new QPushButton[20](labl1);


        int psmstat[5]={0,1,1,1,1};
        int swstat[5]={1,1,1,1,1};
        int SSstat[5]={1,1,1,1,1};
        for(int i=0;i<nsubsys;i++)
        {
          QString str1=" ";
          QString str2=" ";
          str1=SS[i].ssid;
                str2=SS[i].ssname;
```

```
    btn1->move(x,y);
    btn1->resize(100,30);
    btn1->setText(str2);
    QToolTip::add(btn1,str1);
    x+=200;
  } //end of subsys fail


//Slot failure
 int d=0,k;
 int x1=250,y1=200;
 int j;
for(int i=0;i<nsubsys;i++)
 {
      for(j=d,k=0;SS[i].component[k]!='\0';j++,k++)
         {
                 QString str2=SS[i].component[k];
           QPushButton *btn2=&pcbbt[j];
                 btn2->resize(20,10);
           btn2->move(x1+150,y1);

           for(int r=0;r<nPSM;r++)
           {
               if(psmstat[r]==0)
               {
                for(int t=0;P[r].component[t]!='\0';t++)
                {
                     if(P[r].component[t]==SS[i].component[k])
                     {
                       SSstat[i]=0;
                       btn2->setFlat(true);
                       btn2->setPaletteBackgroundColor("red");
                     }
                }
                }
            }

           for(int p=0;p<nswitch;p++)
            {
                if(swstat[p]==0)
                {
                for(int q=0;SW[p].component[q]!='\0';q++)
                 {
                      if(SW[p].component[q]==SS[i].component[k])
                        {
                        SSstat[i]=0;
                        btn2->setFlat(true);
                        btn2->setPaletteBackgroundColor("red");
                        }
                         }
                }
                   }

           for(int i=0;i<5;i++)
            {
                 if(SSstat[i]==0)
                 {
```

```
                    QPushButton *btn1=&ssbt[i];
                    btn1->setFlat(true);
                    btn1->setPaletteBackgroundColor("red");
                   }
                }
                QToolTip::add(btn2,str2);
                x1+=35;

              }
            d+=k;
          x1+=55;
       } //end of slot fail

       //PSM and Switch Failure
      int x2=500,y2=300;
     for(int i=0;i<nPSM;i++)
      {
          QString str3=P[i].PSM_id;
          QpushButton *btn3=&psmbt[i];
          btn3->resize(20,10);
                btn3->move(x2,y2);
           if(psmstat[i]==0)
               btn3->setPaletteBackgroundColor("red");
          QtoolTip::add(btn3,str3);
          x2+=35;
       }

      int x3=x2+20,y3=300;
      for(int i=0;i<nswitch;i++)
       {
          Qstring str4=SW[i].switch_id;
          QpushButton *btn4=&swbt[i];
          btn4->resize(20,10);
                  btn4->move(x3,y3);
          if(swstat[i]==0)
               btn4->setPaletteBackgroundColor("red");
          QtoolTip::add(btn4,str4);
          x3+=35;
        }

         labl1->show();

}
```

Code for Fault Tree

```
void SpecForm::drawTree()
{
 QLabel *labl =new QLabel(0);
 labl->setFixedSize(1000,700);
 labl->setFrameStyle( QFrame::Panel | QFrame::Raised );
 labl->setPaletteBackgroundColor("black") ;
    int x=400,y=100;
    QPushButton *mainbt =new QPushButton(labl);
    mainbt->resize(100,30);
    mainbt->move(500,50);
    mainbt->setText("System Fail");
    mainbt->setPaletteBackgroundColor("magenta") ;
    QPushButton *ssbt =new QPushButton[10](labl);
    QPushButton *pcbbt =new QPushButton[200](labl);
    QPushButton *psmbt =new QPushButton[20](labl);
    QPushButton *swbt =new QPushButton[20](labl);


    for(int i=0;i<nsubsys;i++)
    {
      QString str1=" ";
      QString str2=" ";
      str1=SS[i].ssid;
            str2=SS[i].ssname;
      QPushButton *btn1=&ssbt[i];
      btn1->move(x,y);
      btn1->resize(100,20);
      btn1->setText(str2);
      btn1->setPaletteBackgroundColor("pink") ;
      QToolTip::add(btn1,str1);
      x+=200;
    } //end of subsys fail
    //PSM and Switch Failure


   //Slot failure
    int d=0,k;
    int x1=250,y1=200;
    int j;
   for(int i=0;i<nsubsys;i++)
    {
         for(j=d,k=0;SS[i].component[k]!='\0';j++,k++)
            {
                    QString str2=SS[i].component[k];
              QPushButton *btn2=&pcbbt[j];

              btn2->resize(20,10);
              btn2->move(x1+150,y1);
              btn2->setPaletteBackgroundColor("light blue") ;
```

```
            QToolTip::add(btn2,str2);
            x1+=35;

          }
        d+=k;
      x1+=55;
  } //end of slot fail

 int x2=500,y2=300;
for(int i=0;i<nPSM;i++)
 {
     QString str3=P[i].PSM_id;
     QPushButton *btn3=&psmbt[i];
     btn3->resize(20,10);
            btn3->move(x2,y2);
     btn3->setPaletteBackgroundColor("orange") ;
     QToolTip::add(btn3,str3);
     x2+=35;
 }

 int x3=x2+20,y3=300;
 for(int i=0;i<nswitch;i++)
  {
      QString str4=SW[i].switch_id;
      QPushButton *btn4=&swbt[i];
      btn4->resize(20,10);
             btn4->move(x3,y3);
      btn4->setPaletteBackgroundColor("yellow") ;
      QToolTip::add(btn4,str4);
      x3+=35;
   }


      labl->show();
 }


 void SpecForm::drawFT()
 {
  QLabel *labl1 =new QLabel(0);
  labl1->setFixedSize(1000,700);
  labl1->setFrameStyle( QFrame::Panel | QFrame::Raised );
  labl1->setPaletteBackgroundColor("black") ;
     int x=400,y=100;
     QPushButton *mainbt =new QPushButton(labl1);
     mainbt->resize(100,30);
     mainbt->move(500,50);//100
     mainbt->setText("System Fail");
     mainbt->setPaletteBackgroundColor("red") ;
     QPushButton *ssbt =new QPushButton[10](labl1);
     QPushButton *pcbbt =new QPushButton[200](labl1);
     QPushButton *psmbt =new QPushButton[20](labl1);
     QPushButton *swbt =new QPushButton[20](labl1);
```

```
int psmstat[5]={0,1,1,1,1};
int swstat[5]={1,1,1,1,1};
int SSstat[5]={1,1,1,1,1};
for(int i=0;i<nsubsys;i++)
{
  QString str1=" ";
  QString str2=" ";
  str1=SS[i].ssid;
          str2=SS[i].ssname;
  QPushButton *btn1=&ssbt[i];
  btn1->move(x,y);
  btn1->resize(100,30);
  btn1->setText(str2);
  QToolTip::add(btn1,str1);
  x+=200;
 } //end of subsys fail

//Slot failure
 int d=0,k;
 int x1=250,y1=200;
 int j;
for(int i=0;i<nsubsys;i++)
 {
      for(j=d,k=0;SS[i].component[k]!='\0';j++,k++)
         {
                  QString str2=SS[i].component[k];
           QPushButton *btn2=&pcbbt[j];
                  btn2->resize(20,10);
           btn2->move(x1+150,y1);

           for(int r=0;r<nPSM;r++)
           {
               if(psmstat[r]==0)
               {
                for(int t=0;P[r].component[t]!='\0';t++)
                {
                     if(P[r].component[t]==SS[i].component[k])
                     {
                       SSstat[i]=0;
                       btn2->setFlat(true);
                       btn2->setPaletteBackgroundColor("red");
                      }
                }
                }
            }

           for(int p=0;p<nswitch;p++)
            {
                if(swstat[p]==0)
                {
                for(int q=0;SW[p].component[q]!='\0';q++)
                 {
                      if(SW[p].component[q]==SS[i].component[k])
                        {
                        SSstat[i]=0;
                        btn2->setFlat(true);
```

```
                        btn2->setPaletteBackgroundColor("red");
                        }
                         }
                    }
                       }

          for(int i=0;i<5;i++)
           {
                if(SSstat[i]==0)
                {
                 QPushButton *btn1=&ssbt[i];
                 btn1->setFlat(true);
                 btn1->setPaletteBackgroundColor("red");
                }
              }
              QToolTip::add(btn2,str2);
              x1+=35;

            }
         d+=k;
       x1+=55;
  } //end of slot fail

 int x2=500,y2=300;
for(int i=0;i<nPSM;i++)
 {
     QString str3=P[i].PSM_id;
     QPushButton *btn3=&psmbt[i];
     btn3->resize(20,10);
             btn3->move(x2,y2);
      if(psmstat[i]==0)
          btn3->setPaletteBackgroundColor("red");
     QToolTip::add(btn3,str3);
     x2+=35;
  }

  int x3=x2+20,y3=300;
  for(int i=0;i<nswitch;i++)
   {
       QString str4=SW[i].switch_id;
       QPushButton *btn4=&swbt[i];
       btn4->resize(20,10);
               btn4->move(x3,y3);
       if(swstat[i]==0)
            btn4->setPaletteBackgroundColor("red");
       QToolTip::add(btn4,str4);
       x3+=35;
    }

     lab11->show();

  }
```

5. 3 Future Scope

This area of project provides large scope for further enhancements and improvements. Below listed are a few suggestions:

- Use database system instead of file system for information storage.
- Include further refined connectivity details in the system layout and fault tree.
- Provide a detailed fault tree including all possible types of faults.
- Enhancement of failure paths to units /sub-modules inside PCBs.

APPENDIX

REFERENCES

[1] **"Target,Estimation,Detection and Tracking"** Martin Hurtado, Jin-Jun Xiao and Arye Nehorai IEEE Transaction Signal Processing Vol.26 pp 42-52 Jan 2009

[2] **"Researches on the measurement of the distribution image of radiated noise using focused beamforming"** HUI Juan HU Dan HUI Junying YIN Jingwei Chinese Journal of Acoustics Vol.27 No.2, 2008

[3] C.G.Capus A.C.Bauks E.Coiras I.tena Ruiz C.J. Smith Y.R.Petillot **"Data correction for visualization and classification of sidescan SONAR imagery"**, published in IET Radar,Sonar and Navigation, 5 June 2007

[4] **"Encyclopedia of Physical Science and Technology"**, Vol.1 A-Arc Robert.A.Meyers 1987

[5] **"Encyclopedia of Ocean Sciences"**, Vol.5, John.H.Steele, 2007

[5] **"The Encyclopedia of Electronics"**, Vol.5, Charles Susskind, 1962

[6] **"Principles of Underwater Sound for Engineers"**, 1 Edition Robert.J.Urick, 1967

[7] **"Failure Analysis and Prevention"** Vol.11 ASM Handbook", 2002

[9] Prof. David Harel,**"Real-Time UML"**, Edition 2 Developing Efficient Objects for Embedded systems

[10] Ghinwa Jalloul "UML by Example"

[11] www.troltech.com

[12] en.wikipedia.org

[13] "Qt Designer", O'Relly publication