White Paper

Implementation of Class B Software

WP001601-0706

ZiLOG Worldwide Headquarters • 532 Race Street • San Jose, CA 95126
Telephone: 408.558.8500 • Fax: 408.558.8300 • www.zilog.com

Abstract

The ZiLOG's Motor Control (MC) Library v1.1.0 is a software library developed following the International Electrotechnical Commission (IEC, www.iec.ch) Class B standard. The MC Library v1.1.0 helps to develop applications that run a sensorless brushless DC (BLDC) motor in constant speed and constant torque mode. The intent of this white paper is to explain the design, coding, testing, and configuration management required to comply with the Class B standard. This white paper provides an example of lift door controller which uses the Class B standard and guidelines to implement the standard.

Introduction to Class B Software

Class B software detects any fault occurring in the appliance and includes exception handling to prevent unintended operation of an appliance. Many household appliances use Class B software. Some common examples are listed below:

- **Microwave oven:** Microwave ovens have door locking mechanism that is interlocked with the operation of the device. The interlock feature ensures that until the door is locked, you cannot run the appliance. Earlier microwave ovens were designed using complex mechanical/electrical system, which had many interlocks. Now these complex interlocking is replaced by microcontroller and sensor.
- **Washing machine:** Washing machines have door locking and water-level sensor interlocked with the operation of the drum through sensor and microcontroller. When you try to operate the washing machine, it checks if the door is locked and adequate water level is present in the machine. Only when both these criteria are satisfied, the machine starts operating. After the washing is complete, the system ensures that the door does not open until the drum has stopped completely and all the water has been drained out from the drum.

Nowadays, household appliances usually inhibit high power inside it and uses this high power to make things work faster. Some of the examples are listed below:

- High speed—Spinning drum in washing machine and lift doors.
- High voltage—Internal power supply for klystron tube in microwave oven, photocopying machine, and laser printer.
- Radiation—Microwave and laser in CD recording.

IEC publishes standards for electronic industries and related technologies. Details on Class B standard and their implementation procedure is obtained from the following standards:

- IEC 60335-1/ A-1
  Household and similar electrical appliances—Safety
  Annex—R Part 1: General requirements.
- IEC 60730-1 Copil:20031
  Automatic electrical controls for household and Annex—H
  Similar use—Part 1: General requirements

General Guidelines for Developing Class B Software

The initial analysis for developing Class B software is to identify the possible faults that occurs in the system. An effective analysis tool for evaluating faults in a system is the fault tree analysis. For detailed information on fault tree analysis, refer the following link.

http://www.uscg.mil/hq/gm/risk/e-guidelines/rbdm/html/vol3/00/v3-00.htm

Lift door controller is considered as an example for carrying out fault tree analysis to find out the various faults and the ways to mitigate them.

Initially the normal operation of the lift door controller without Class B features is explained in the following section. Figure 1 illustrates the lift door controller block diagram. Figure 2 illustrates the flow chart for the lift door controller without Class B features.

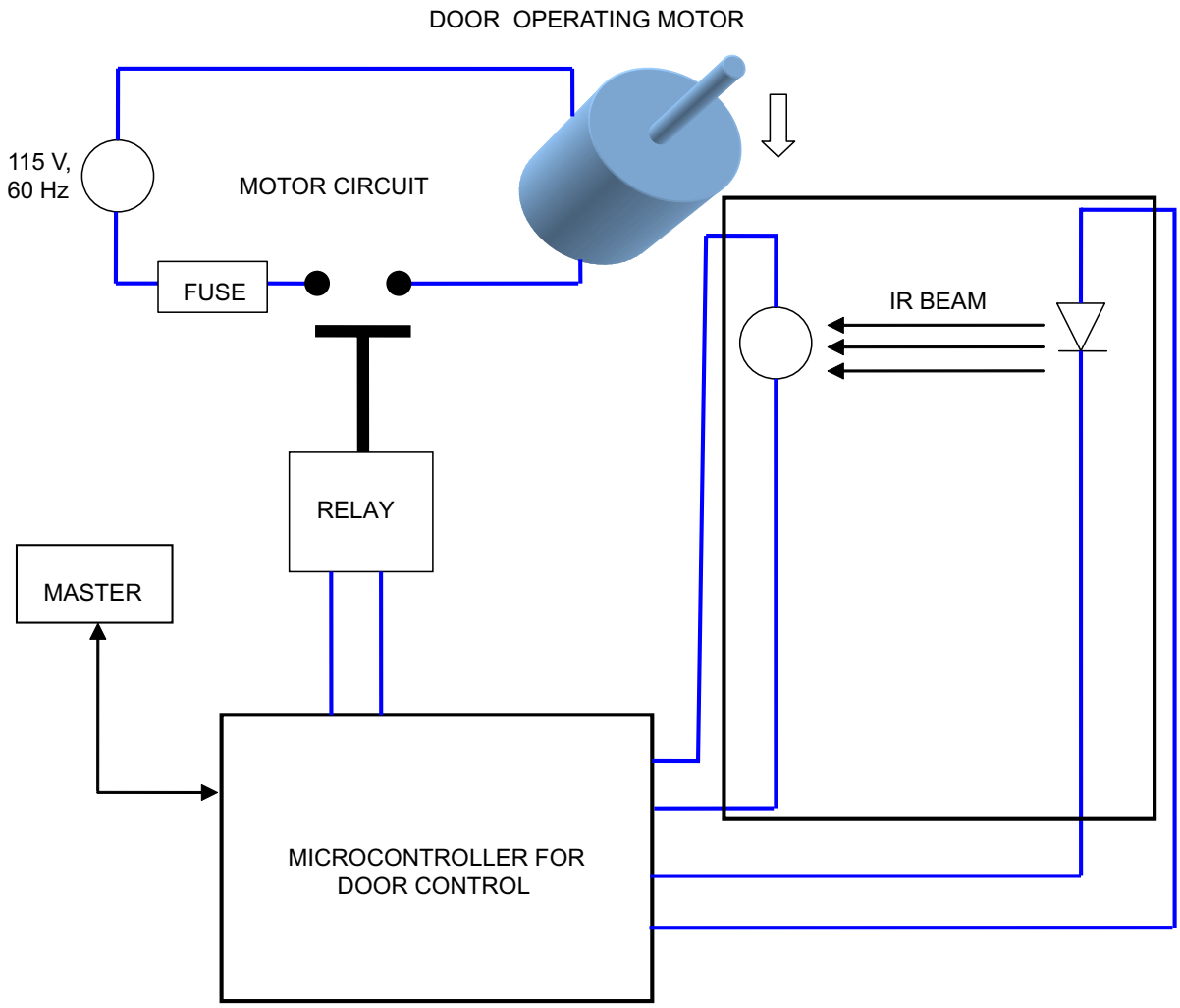

Figure 1. Lift Door Controller Block Diagram

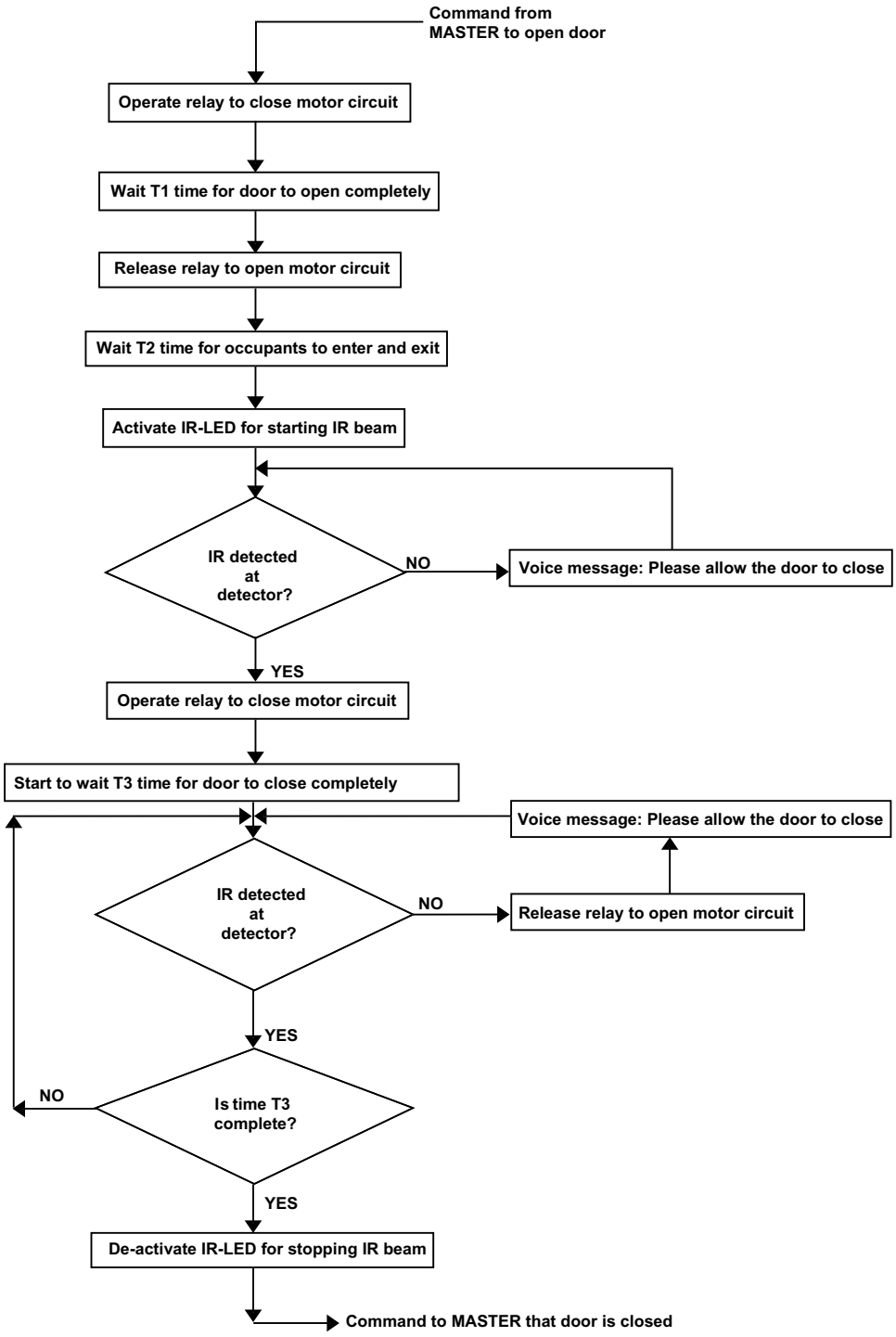

Figure 2. Flow Chart for Normal Operation of Lift Door Controller

Pseudocode for Lift Door Controller

The following list provides the pseudocode for normal operation of lift door controller without Class B features:

1. Wait for command from MASTER to open the door.
2. Send command to relay to close motor circuit.
3. Wait for time T1, for the door to open completely.
4. Send command to relay to open the motor circuit.
5. Wait for time T2, for occupant to enter and exit.
6. Start the IR beam through IR-LED.
7. Check for IR beam detection at the detector.
8. If IR beam is not detected, then send voice message through a speaker as 'Please allow the door to close'. The IR loop continues till IR beam is detected.
9. If IR beam is detected, then send command to relay to close the motor circuit.
10. Wait for a time T3, for the door to close completely.
11. Check for IR beam detection.
12. Stop the IR beam through IR-LED.
13. Send command to MASTER that door is closed.

Software Coding for Normal Operation of Lift Door Controller

```
/*****************************************************************************

* Code for lift door controller without Class B features

*****************************************************************************/
  while(!(PORTA & MASTER_COMMAND))
  {
      /* wait for command input from MASTER to open door */
  }
      /* command received from MASTER to open door */

  PORTA |= RELAY_ON;
      /* relay energized to close motor circuit */

  for(i = 0; i <= T1; i++)
```

```
{
    /* wait for time T1, for the door to completely open */
}

PORTA &= ~RELAY_ON;
    /* relay de-energized to open motor circuit */

for(i = 0; i <= T2; i++)
{
    /* wait for time T2, for the occupants to enter or exit
    */
}

/* time to close the door. Check if the door is clear of
occupants */
PORTA |= LED_ON;                    /* start IR beam */

/* if IR not detected at sensor, announce through speaker
to clear the door */
while(!(PORTA & IR_DETECTED))     /* IR not detected */
{
    send_voice();
    /* function for speaker to announce 'Please allow the
    door to close'*/
}
/* IR detected by sensor and hence door is closed */

PORTA |= RELAY_ON;
/* relay energized to close motor circuit */

/* wait for time T3, for the door to get closed */
for(i = 0; i <= T3; i++)
{
    /* During waiting time T3, check if IR beam is
    interrupted */
    while(!(PORTA & IR_DETECTED)) /* IR not detected */
    {

        PORTA &= ~RELAY_ON;
        /* relay de-energized to open motor circuit */
        send_voice();
        /* function for speaker to announce 'Please allow
        the door to close'*/
    }
```

```
}
/* door is closed now */
PORTA &= ~LED_ON;                    /* stop IR beam */

/* send command to MASTER that door is closed */
```

Fault Tree Analysis for Lift Door Controller

The following list describes the various steps involved in the fault tree analysis for lift door controller.

1. Define the system of interest
   - Intended function—To control the opening and closing of the lift door without causing any inconvenience to occupants.
   - Physical boundaries—Communication with MASTER.
   - Analytical boundaries—Ignore wiring faults; ignore power supply failure.
   - Initial conditions—Relay open, motor not operating and microcontroller waiting for command from MASTER to open the door.
2. Define the top event for analysis
   - Door fails to open completely.
   - Door closes while the occupants are still at door.
   - Door fails to close completely.
3. Define the treetop structure

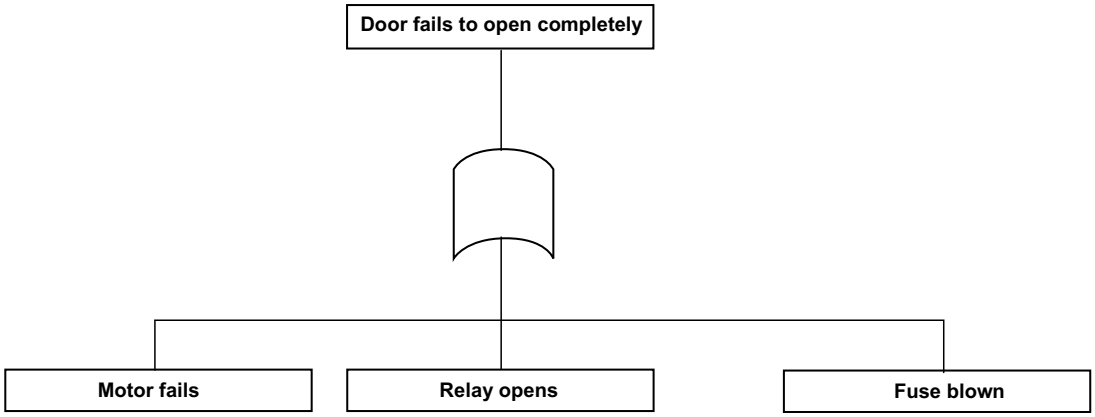

Figure 3. Top Level Fault Analysis-1(Tree Top Structure 1)

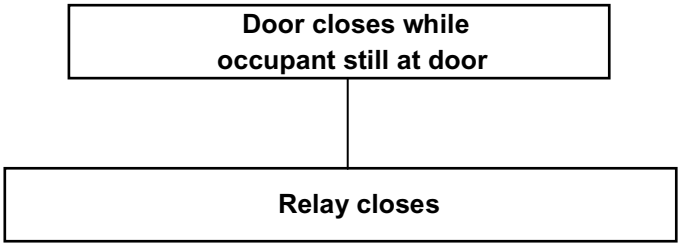

Figure 4. Top Level Fault Analysis-2 (Tree Top Structure 2)

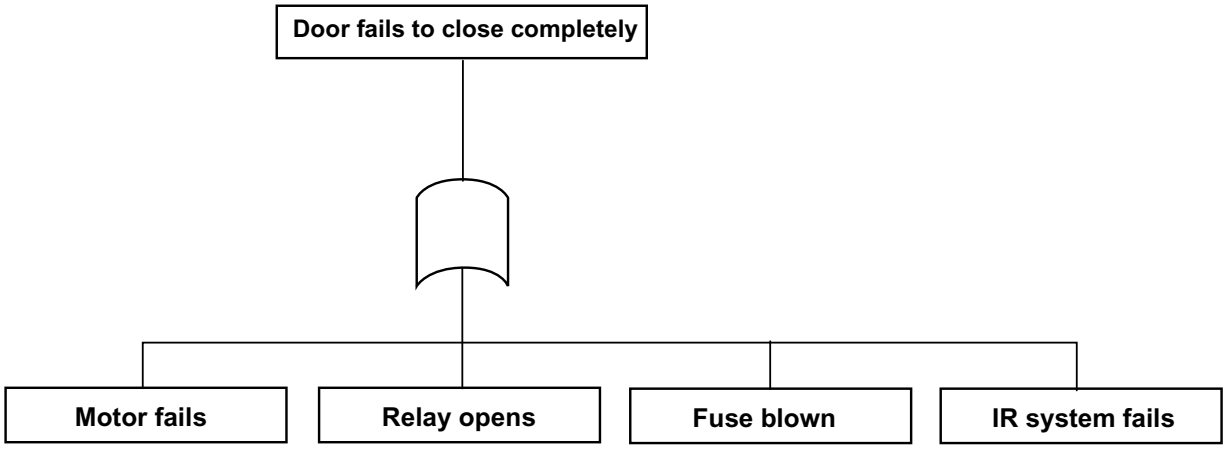

Figure 5. Top Level Fault Analysis-3 (Tree Top Structure 3)

Figure 6 through Figure 8 illustrates the detailed fault tree analysis for the lift door controller.

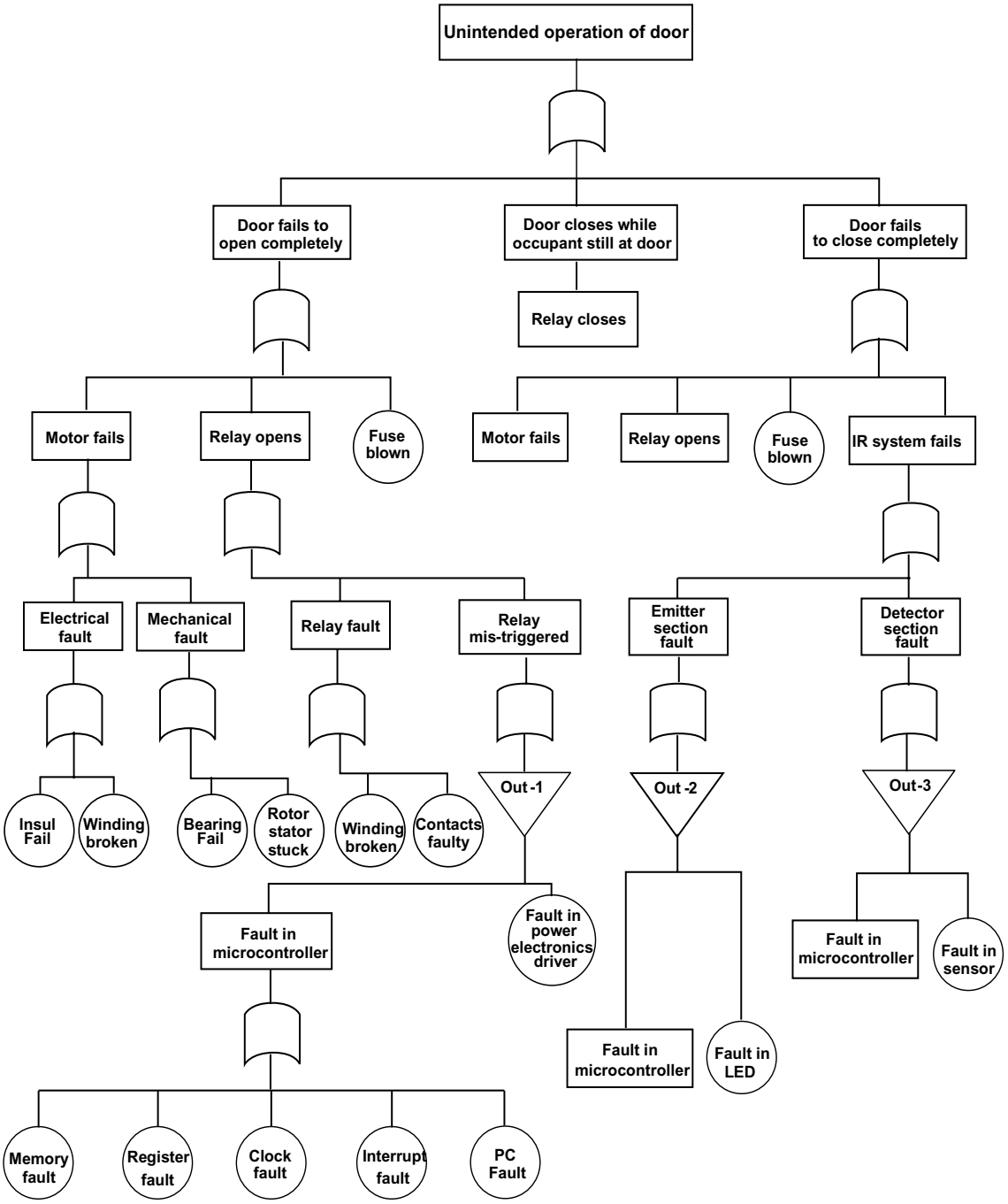

Figure 6. Fault Tree for Entire System

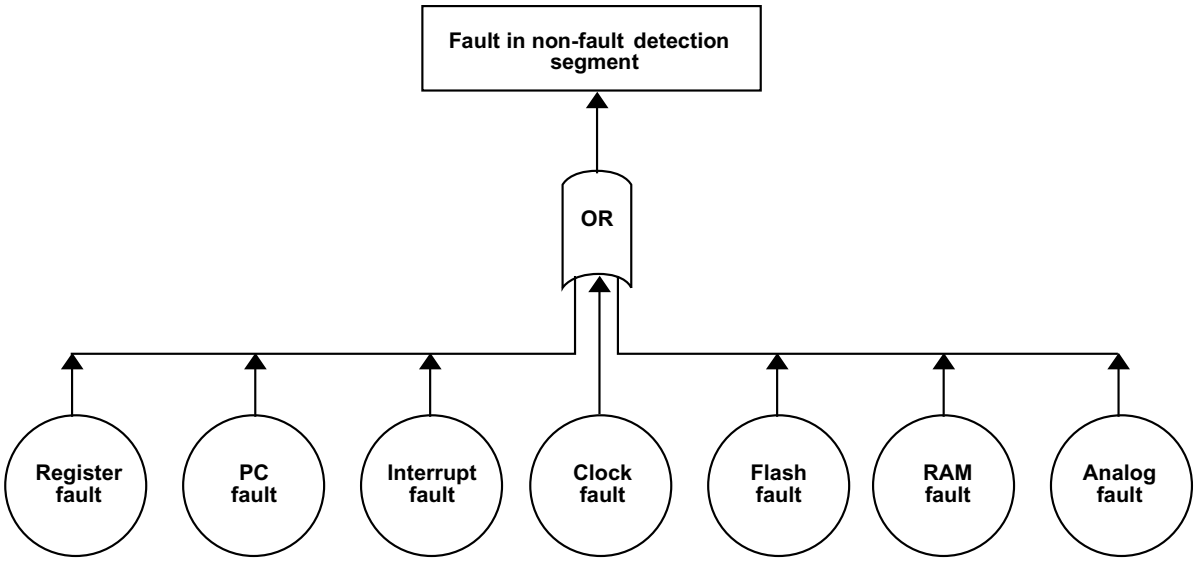

Figure 7. Fault Tree for Non-Fault Detection Segment

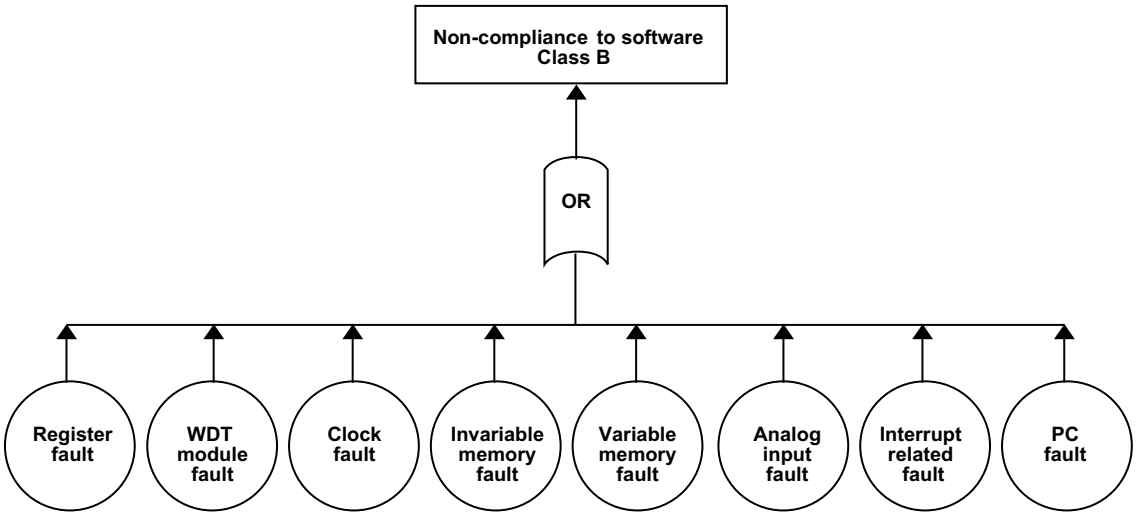

Figure 8. Fault Tree for Fault Detection Segment

From the fault tree analysis, the following faults are identified, which results in an unintended operation of the system:

- Motor insulation failure
- Motor winding broken
- Bearing failure
- Rotor and stator stuck

- Fuse blown
- Relay winding broken
- Fault in relay contact
- Fault in power electronic driver
- Fault in IR-LED
- Fault in IR-sensor
- Memory fault in microcontroller
- Register fault in microcontroller
- Clock fault in microcontroller
- Interrupt fault in microcontroller
- Program counter fault in microcontroller

Acceptable Measures

The following measures detects the occurrence of fault in lift door controller:

- Current sensor (Hall sensor or current transformer or shunt resistor) placed external to the motor circuit senses the flow of current. The output of current sensor applied to the microcontroller. The software code is modified to include feature that check the presence/absence of current in motor circuit.
- Redundant IR-sensor: This feature needs another IR-detector to be placed beside the actual IR-sensor. The software code is modified to check both the actual and redundant sensor; and when both are verified to be true, the control moves to the next statement.
- Word protection with single bit redundancy[1]: This memory test is done periodically during run time to detect any fault occurring in Flash (invariable memory).
- Static memory test[2]: This memory test is done periodically during run time to detect any fault in RAM (variable memory).
- Static memory test: This memory test is done periodically during run time to detect any fault occurring in CPU register.
- Independent time slot monitoring[3]: Used for detecting fault in program counter.

1.A fault/error control technique where a single bit is added to each word in the memory area under test and saved, creating either even parity or odd parity. As each word is read, a parity check is conducted.

2.A fault/error control technique which is intended to detect only static errors.

4. Primary clock monitor: Used for continuous monitoring of primary clock and generate interrupt in case of failure. If an interrupt is generated, Watchdog timer (WDT) takes control over the execution of code in ISR.

Table 1 summarizes the various faults and acceptable measures for the lift door controller.

Table 1. Fault and Acceptable Measures for Lift Door Controller

No	Component	Sub-component	Fault/Error	Acceptable Measures
1	CPU	Register	Stuck at	Periodic self-test using static memory test.
		Program counter	Stuck at	Independent time slot monitoring.
2	Interrupt handling and execution		No interrupt or too frequent interrupt	Time slot monitoring.
3	Clock		Wrong frequency	Time slot monitoring.
4	Memory	Invariable memory[a]	All single bit faults	Word protection with single bit redundancy.
		Variable memory[b]	DC fault	Word protection with single bit redundancy.
		Addressing (both variable and invariable memory)	Stuck at	Word protection with single bit parity including the address.
5	Input/Output periphery	Analog MUX for current sense	Wrong addressing	Plausibility check[c]
		Digital input from IR-sensor		Plausibility check
6	Motor circuit	Motor	Winding failure	Current sense
			Insulation failure	Current sense
			Bearing failure	Current sense
			Rotor and stator stuck	Current sense
		Fuse blown		Current sense
7	Relay circuit	Relay	Winding failure	Current sense
			Faulty contact	
		Power electronics driver		Current sense

3.A fault/error control technique in which timing devices with an independent time base are periodically triggered in order to monitor the program function and sequence.

Table 1. Fault and Acceptable Measures for Lift Door Controller (Continued)

8	IR circuit	IR LED	Fault in LED	Redundant sensor
		IR sensor	Fault in sensor	Redundant sensor

a. Memory range in a processor system containing data which is not intended to vary during program execution.
b. Memory range in a processor system containing data which is intended to vary during program execution.
c. A fault/error control technique in which program execution, inputs or outputs are checked for inadmissible program sequence, timing or data.

As a requirement for Class B standard[4],the fault detection and acceptable measure for sub-system inside the microcontroller must be complying with the measures listed in IEC standard.

Based on the functions performed by software code, the software code is divided into two segments as:

- Fault detection segment: The fault detection segment detects and mitigates each fault. The mitigation of each fault is carried out by completely shutting down the system or by reducing the intensity of the cause of fault.
- Non-fault detection segment: The non-fault detection segment carries out other function to take care of proper functioning of the apparatus, like, for carrying out the function of running the motor.

Note: If an exception occurs in the fault detection segment of software without proper handling, then the software will no longer comply to Class B standard. Exceptions like this must be documented in manuals for the customer to know about them.

Control Response for Faults

Once the faults are detected, the designer needs to decide the way in which the system must respond to a detected fault. The control response depends on the application and the way the specification wants the system to perform.

Example—when a fault in IR circuit is detected, a designer might want that only the lift to be stopped, while another designer might want an announcement through the speaker for the specific fault that is detected.

Table 2 lists some of the probable responses that occurs in practical application/requirement.

4.IEC60730-1/A1. Table-H.11.12.7 provides guidelines for identifying a list of faults and acceptable measures to be included in the software code.

Table 2. Control Response for Faults

Fault	Control Response for Fault
Fault in register	Stop the lift and announce through speaker as request for maintenance.
Fault in program counter	
Faulty interrupt	
Faulty clock frequency	
Single bit fault in Flash memory	
Fault in RAM	
Fault in motor circuit (winding broken, insulation short, and fuse blown)	Stop the lift and announce through speaker as request for maintenance on motor circuit.
Fault in relay circuit or power electronics driver	Stop the lift and announce through speaker as request for maintenance on relay circuit.
Fault in IR circuit	Stop the lift and announce through speaker as request for maintenance on sensor and emitter.

Software Coding for Non-Fault Detection Segment

```
/*****************************************************/
The following Code is the modified version of existing
normal code for lift door controller for detecting and
responding to the fault.
/*****************************************************/
while(!(PORTA & MASTER_COMMAND))
{
    /* wait for command input from MASTER to open door */
}
/* command received from MASTER to open door */

PORTA |= RELAY_ON;
/* relay energized to close motor circuit */

for(i = 0; i <= T1; i++)
{
    check_current_ON();
    /* check for current in motor circuit, it helps in
    detecting if the motor stops in between and the door
    fails to open completely */
```

```
    /* wait for time T1, for the door to open completely*/
}

PORTA &= ~RELAY_ON;
/* relay de-energized to open motor circuit */

for(i = 0; i <= T2; i++)
{
    check_current_OFF();
    /* check that no current is flowing in the motor
    circuit, so that motor does not start running while the
    occupants are at the door */
    /* wait for time T2, for the occupants to come out or
    get in */
}

/* time to close the door. Check if the door is clear of
occupants */
PORTA |= LED_ON;                    /* start IR beam */

while(!(PORTA & IR_DETECTED))
/* IR not detected by primary sensor */
    {
    if(!(PORTA & IR_REDUNDANT_DETECTED))
    /* IR not detected by redundant sensor also. It is true
    that someone is at the door. Hence announce to
    occupants to clear the door */
        {
        send_voice();
        /* function for speaker to announce 'Please allow
        the door to close' */
        }
        else
        /* if redundant sensor detect IR, means primary
        sensor is faulty */
        {
        send_voice();
        /* announce that 'IR_detector is faulty' */
        }
    }
/* IR detected by sensor and hence door is closed */

PORTA |= RELAY_ON;
/* relay energized to close motor circuit */
```

```
/* wait for time T3, for the door to get closed */
for(i = 0; i <= T3; i++)
{
    check_current_ON();
    /* check for current in motor circuit, it helps in
    detecting if the motor stops in between and the door
    fails to close completely */
    /* During waiting time T3, check if IR beam is
    interrupted */
    while(!(PORTA & IR_DETECTED)) /* IR not detected */
    {
    PORTA &= ~RELAY_ON;
    /* relay de-energized to open motor circuit */
    send_voice();
    /* function for speaker to announce 'Please allow the
    door to close' */
    }
}
/* door is closed now */
PORTA &= ~LED_ON;                  /* stop IR beam */

/* send command to MASTER that door is closed */
```

Software Coding for Fault Detection Segment

```
/*********************************************************
* This function checks if current is flowing in the motor
circuit and is within the limits. This function is to be
used during opening or closing lift door.
* This code will prevent unintended working of lift door.
If any fault occurs in
* 1. Motor, 2. Fuse 3. Relay circuit, 4. Power electronics
driver for relay coil, the working of door motor is
interrupted.
*********************************************************/
unsigned char check_current_ON(void)
{
while (ADCCTL0 & 0x80);
/* Wait for end of conversion */
ADCCTL0 = (0x30 | CURRENTSENSE);
/* Enable ADC channel for bus voltage */
ADCCTL0 |= 0x80;                   /* Start conversion */
```

```
while (ADCCTL0 & 0x80);
/* Wait for end of conversion */
motor_current = ADCD_H;
/* load current value from ADC-register to another */
if((motor_current > MIN_CURRENT) && (motor_current <=
MAX_CURRENT))
{
return(0x00);          /* 0x00 implies SUCCESS */
}
else
{
/* Emergency Stop and sound alarm accordingly */
}
}
/*********************************************************
* This function checks if no current is flowing in the
motor circuit. This function ensures that the motor does
not start running when the occupants are at the lift door.
* This code will prevent the fault in relay contacts
shorted by external conducting material, when the relay is
still de-energized.
*********************************************************/
unsigned char check_current_OFF(void)
{
while (ADCCTL0 & 0x80);
/* Wait for end of conversion */
ADCCTL0 = (0x30 | CURRENTSENSE);
/* Enable ADC channel for bus voltage */
ADCCTL0 |= 0x80;
/* Start conversion */
while (ADCCTL0 & 0x80);
/* Wait for end of conversion */
motor_current = ADCD_H;
/* load value of current from ADC-register to another */
if((motor_current == 0x00) && (motor_current <
MIN_CURRENT))
{
return(0x00);          /* 0x00 implies SUCCESS */
}
else
{
/* Emergency Stop and sound alarm accordingly */
}
}
```

Software Coding for Fault Detection Segment

```
/*********************************************************
* The function runs periodically through a timer interrupt
to detect fault in memory and CPU register.
*********************************************************/
#pragma interrupt
timer0_interrupt_service_routine(void)
{
static unsigned char step = 0x00;
    if(step == 0x00)
    {
    FLASH_memory_test();
    /* check appendix for code implementation */
    }
    if(step == 0x00)
    {
    RAM_memory_test();
    /* check appendix for code implementation */
    }
    if(step == 0x00)
    {
    register_test();
    /* check appendix for code implementation */
    }
}

/*********************************************************
* Additionally initialize WDT, which takes care of
detection of faults like clock fault, interrupt fault,
program counter fault etc. See appendix for examples on
detection of these faults.
*********************************************************/
void init_wdt(void)
{
    WDTH = 0x55;        /* unlock sequence - 1 */
    WDTH = 0xAA;        /* unlock sequence - 2 */
    WDTH = 0x03;        /* load 0x0300 = 77 ms */
    WDTL = 0x00;
}
```

Software Testing

Like any standard software development process, the development of Class B compliant software also flows through the testing phase. It comprise of:

1. Unit testing
2. Integration testing
3. System testing

Documents related to software testing

The following documents must be provided to the standard agency during certification as a part of testing/validation phase of software:

- **Unit test plan (UTP)—**The UTP must cover the test case ID, description of the test case (which particular module/feature of software will be tested using this test case ID), test topology with details of hardware and software settings, environment and external dependencies, procedure for conducting the test, expected output, and pass/fail criteria. In addition, the test plan specifies the point from where the test must be resumed if a test case fails in between. The testing must cover all the possible modes of operation; memory models etc., and each test case must be traceable to the software design specification through a requirement traceable matrix.
- **Unit test report (UTR)—**The UTR must record the outcome of each test cases, date and time of testing, person responsible for testing, and remark as the test has passed or failed.
- **System test plan (STP)—**The content of STP is similar to UTP except the test cases must be designed keeping in mind about the final application and how individual modules interact with each other to make the final system.
- **System test report (STR)—**The format of STR is similar to UTR while the content is based on the test cases as planned in the STP.

Tests to be carried out as a part of integration testing. Generating of related documents like integration test plan and integration test report is skipped for smaller applications with limited number of individual modules, where system test itself handles all the cases of interconnection of modules to form the entire application.

Unit testing

During unit testing, the features of Class B are tested. The faults identified through the fault tree analysis are listed and during each test, the fault is simulated in the system. Then the system response is recorded. If all the faults are detected and

there is no unintended operation of system due to the fault, then the implementation of Class B feature is declared to be passed. The following section explains the unit test plan for testing each of the Class B features of software.

Test Case ID	UT_Negative_001
Description	This is a negative test case to detect the fault in motor circuit when the motor stops half the way while opening the door.
Test topology	Test set up is a simple connection of a switch in motor circuit by which the motor circuit is broken (simulating a fault).
Precondition	None
Inputs	No input parameters.
Procedure	When the door is opening and is half the way open, break the motor circuit by opening the switch.
Outputs	The door stops and specific message is announced through the speaker that ‘motor circuit is broken and need maintenance’.
Remarks if any	

Test Case ID	UT_Negative_002
Description	This is a negative test case used to detect fault in motor circuit when the motor starts and door starts closing while occupants are still at door.
Test topology	Test set up is a simple connection of a switch across the relay contacts in motor circuit by which the motor circuit is closed (simulating a fault).
Precondition	None
Inputs	No input parameters.
Procedure	After the door opens and the occupants are suppose to move in or out of lift, short the relay contact.
Outputs	The door does not close and specific message is announced through the speaker ‘that motor circuit has a fault and needs maintenance’.
Remarks if any	

Test Case ID	UT_Negative_003

Description	This is a negative test case used to detect the fault in IR detection circuit when the sensor is faulty.
Test topology	Access to the IR sensor is required.
Precondition	None
Inputs	No input parameters.
Procedure	After the door opens and the occupants are suppose to move in or out of lift, interrupt the IR beam to sensor by an opaque object.
Outputs	The door does not close and specific message is announced through speaker that ‘IR sensor is faulty and needs maintenance’.
Remarks if any	

Test Case ID	UT_Negative_004
Description	This is a negative test case used to test the situation when the IR beam is blocked by occupant in lift.
Test topology	Access to the both IR sensor is required.
Precondition	None
Inputs	No input parameters.
Procedure	After the door opens and the occupants are suppose to move in or out of lift, interrupt both the IR beam to sensor by an opaque object.
Outputs	The door does not close and specific message is announced through speakers that ‘please allow the door to close’.
Remarks if any	

Test Case ID	UT_Negative_005
Description	This is a negative test case used to detect the fault in motor circuit when the motor stops half the way while closing the door.
Test topology	Test set up is a simple connection of a switch in motor circuit by which the motor circuit is broken (simulating a fault).
Precondition	None
Inputs	No input parameters.

Procedure	When the door is closing and is half the way close, break the motor circuit by opening the switch.
Outputs	The door stops but specific message is announced through the speaker that ‘motor circuit is broken and needs maintenance’.
Remarks if any	

Test Case ID	UT_Negative_006
Description	This is a negative test case used to detect fault in RAM of microcontroller.
Test topology	Test set up target board connected to PC through the debugger.
Precondition	None
Inputs	No input parameters.
Procedure	Simulate a static fault in RAM of microcontroller through a separate software code.
Outputs	The system stops.
Remarks if any	

Test Case IDUT_Negative_007

Description	This is a negative test case used to detect fault in Flash of microcontroller.
Test topology	Test set up target board connected to PC through the debugger.
Precondition	None
Inputs	No input parameters.
Procedure	Simulate a single bit fault in Flash of microcontroller through a separate software code.
Outputs	The system stops.
Remarks if any	

Test Case IDUT_Negative_008

Description	This is a negative test case used to detect fault in CPU register of microcontroller.
Test topology	Test set up target board connected to PC through the debugger.

Precondition	None
Inputs	No input parameters.
Procedure	Simulate a static fault in register of microcontroller through a separate software code.
Outputs	The system stops.
Remarks if any	

Test Case ID	UT_Negative_009
Description	This is a negative test case used to detect fault in clock of microcontroller.
Test topology	Test set up target board connected to PC through the debugger.
Precondition	None
Inputs	No input parameters.
Procedure	Simulate a clock fault in microcontroller by shorting the XIN pin to ground through a 1 K resistor.
Outputs	The system stops.
Remarks if any	

Test Case ID	UT_Negative_010
Description	This is a negative test case used to detect fault in program counter of microcontroller.
Test topology	Test set up target board connected to PC through the debugger.
Precondition	None
Inputs	No input parameters.
Procedure	Simulate a program counter fault in microcontroller by jumping to an unknown location in memory.
Outputs	The system stops (as the WDT does not get a chance to refresh and hence resets the microcontroller).
Remarks if any	

System Testing

During system testing, various modules are combined to provide the function of the overall system. The plan for system testing is dependent on the application and must also have test cases to test the performance of fault detection module of software.

Configuration Management of Software

The following list provides the procedure for software configuration management:

1. **Requirement change management:** The requirement of each feature (like detection of different faults) must be document in the form of a requirement traceability matrix. Whenever there is a change in the requirement or any feature is enhanced, there must be a proper documentation regarding the reason for change, person responsible for updating the code to implement the change, version number from which the change is implemented in code, test cases and test results for testing the changed requirement.
2. **Record documenting problems, changes made and phase of the life cycle:** Any issue of the software (or specific feature of software) must be corrected by making record of changes made. The change starts with writing a change-request (CR) on the version control system, which states the symptoms of problem. The change process must follow the guidelines as specified in the requirement change management. Once the problem is resolved, the CR is closed.
3. **Historical traceability:** The traceability of each feature is possible from design phase to testing phase through a requirement traceability matrix. Once the software is released, the traceability must be possible by implementing proper documentation of changes made, as specified in the above points.
4. **Configuration identification scheme:** Whenever a change is employed and newer version of software is released, it must be carried out by using proper base-line and labeling scheme, which must follow a format as decided and recorded in quality documents.
5. **Methods and activities used to formally control receipt, storage, handling and release of configurable items:** Software to be configured must have proper version control with adequate transparency on storage of each version, testing and validation before release etc. There must be proper documentation for each step to provide witness during the audit and review process.

Outlined Steps for Class B Certification

After the completion of software testing, the software is audited by the agencies to obtain Class B certification. Some of the agencies capable of performing Class B certification test include:

- Underwriter's Laboratory (UL), Northbrook, IL, United states (Anura.S.Fernando@us.ul.com, Francis.G.Ladonne@us.ul.com; Joseph.S.Antony@us.ul.com).
- KEMA Quality B.V, The Netherlands (Nijhuis, Eddie [Eddie.Nijhuis@kema.com]).
- LCIE S.A. France (michel.brenon@lcie.fr).

Figure 9 displays the general steps of UL certification process.

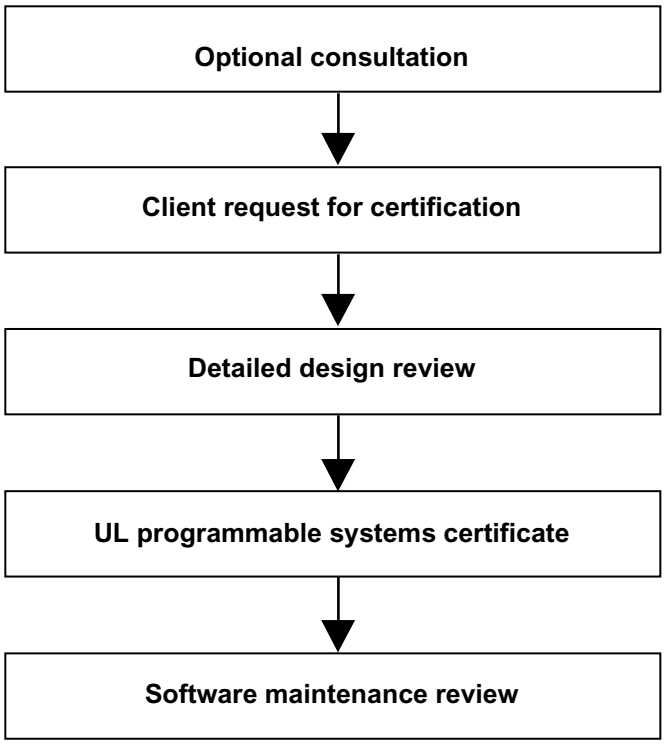

Figure 9. UL Certification Process

Optional Consultation

Optional consultation includes the following steps:

1. Informational seminar on relevant standards.
2. Assessment of preparedness (Initial review).
3. Design solutions for compliance.
4. Development of maintenance program.

Client Request for Certification

Client request for certification includes the following steps:

1. Design information provided to UL.
2. Identification of applicable standards.

Detailed Design Review

Detailed design review includes the following steps:

1. Documentation review.
2. Process walkthrough.
3. UL witness client testing of programmable system.
4. Review software and electronic system for compliance.

UL Programmable Systems Certificate

UL issues programmable system certification to specified US and international standards to complement UL listing or recognition.

Software Maintenance Review

Periodic assessments of modification to software and programmable components.

ReCertification

When software is designed to be Class B compliant, the fault detection segment is protected from user alteration. This implies that if the software segment is altered, then the entire software has to go through a process of recertification. The same rule applies for MC Library v1.1.0.

Developing Class B Compliant Software Using MC Library v1.1.0

The following steps provide instructions for developing Class B compliant software using MC Library v1.1.0:

1. `MCConfigLib()`—When this API is called, it performs a static memory test on RAM, CPU registers, writes redundant bits for Flash memory and initializes WDT for an overflow period of 77 ms.
2. /***Call other APIs to initialize modules like ADC, Timer-0 ***/.
3. `MCInitPwm()`—When this API is called, it initializes the PWM module along with the interrupt service routine for PWM. The PWM-ISR carries out the tasks for fault detection during run-time.
4. `EI()`—The interrupt is globally enabled.

After completing the above steps, the motor starts functioning and the PWM-ISR periodically runs each of the tests for fault detection.

The fault in clock frequency is detected by a special hardware module: Clock failure detection. Whenever a fault in clock frequency is detected, the primary clock source is switched to WDT clock source and an interrupt is generated. The code inside the interrupt service routine then stops the motor.

From the above explanation, it is clear that developing of Class B compliant software using MC Library v1.1.0 is very simple and straight forward. Since most of the fault detection task runs in back ground, you do not notice a difference in code. Some APIs are defined for detecting faults and their location in the library.

- `MCGetLastError()`—Retrieves the fault detected in the system.
- `MCGetLastErrorLocation()`—Retrieves the memory location where fault is detected in the system (only for RAM and FLASH faults).
- `MCCheckInterrupt()`—Checks if interrupt is disabled globally and stops the motor accordingly.

These APIs are used to know more about the fault during run-time.

The MC Library v1.1.0 falls under the category of *Off the shelf software*, which is used to run a motor along with the fault detection feature of Class B directly after purchasing it. Advantages of this feature include:

- Understanding the complexity of motor control and how to make it Class B compliant is not required.
- Provides a quick turn around time for development of final product.
- Refer to the white paper/manuals provided with the software library and develop your own Class B compliant application with ease.
- Certification process will be much easier than developing an entire code, as the code for motor control is already certified through MC Library v1.1.0.

Even with the *Off the shelf software* feature of MC Library v1.1.0, you must understand that just using the library will not make your software as Class B compliant. Because during designing/testing MC Library v1.1.0, details regarding user application were not known. Hence even after you develop your code using MC Library v1.1.0, you have to go for a certification for your own code.

The certification process might be handled differently by each of standard agencies. But basic steps are same before approaching the agencies for certification:

- Proper understanding of fault detection and non-fault detection related software sections.
- Carrying out rigorous risk analysis/fault tree analysis.

- Mitigation of identified faults in software.
- Proper documentation of project schedule, test plans, and test reports.
- Documentation of user manuals and specific documents for Class B standard. Table 3 lists the documents required as a part of Class B standard.
- Carry out unit test and system test.
- Robust version management.

Table 3. Documents Required for Certification

No	Document Name	Content
1	Software sequence documentation	Flow charts explaining the logic of each software routine for detecting specific faults.
2	Program documentation	Pseudocode explaining the logic of each software routine for detecting specific faults.
3	Software fault analysis	Documentation on risk analysis carried out in form of “Fault tree analysis”, for drilling down various faults that occurs in the system.
4	Analytical measure and fault control techniques	How are faults controlled (mitigated) after their detection, so that any unintended operation of the system is prevented.
5	Software fault detection time	The period of time between the occurrence of a fault/error and the initiation by the software of a declared control response.
6	Modular design	Description of modules in software.

Conclusion

The information in this white paper clearly states that, using MC Library v1.1.0 you can effectively develop a motor control solution which is Class B compliant. This can be developed in minimum time and with minimum information on the complexity of the product.

Appendix A—Fault Tree for MC Library v1.1.0

MC Library v1.1.0 is broadly classified into two software segments performing different functionality:

1. **Non-fault detection segment:** Software segment which takes care of normal functioning of motor.
2. **Fault detection segment:** Software segment which periodically checks for any fault occurring in the non-fault detection segment.

The non-fault detection segment contains API, which are called in user application to initialize the modules in controller. They in turn control the working of the motor. Any fault which occurs in the system under non-fault detection software segment results in deviation of operation of motor from the normal behavior.

The fault detection segment contains code, which is meant to detect any fault in the non-fault detection segment of system. This software segment runs in background (during startup or periodically during run-time). The startup test is performed when the `MCConfiglib()` API is called by user application. The periodic test is performed by PWM-interrupt. Any fault occurring in the fault detection segment results in making MC library unable to detect faults occurring in non-fault detection segment. Literally, it means that if a fault occurs in fault detection segment, any or all faults occurring in non-fault detection segment will go undetected and hence make MC library non-compliant to software Class B.

Figure 10 through Figure 12 illustrates the fault tree analysis carried out on MC Library v1.1.0 to detect different faults that might occur.

Figure 10: Fault tree analysis for non-fault detection software segment.

Figure 11: Detailed fault tree analysis for non-fault detection software segment.

Figure 12: Detailed fault tree analysis for fault detection software segment.

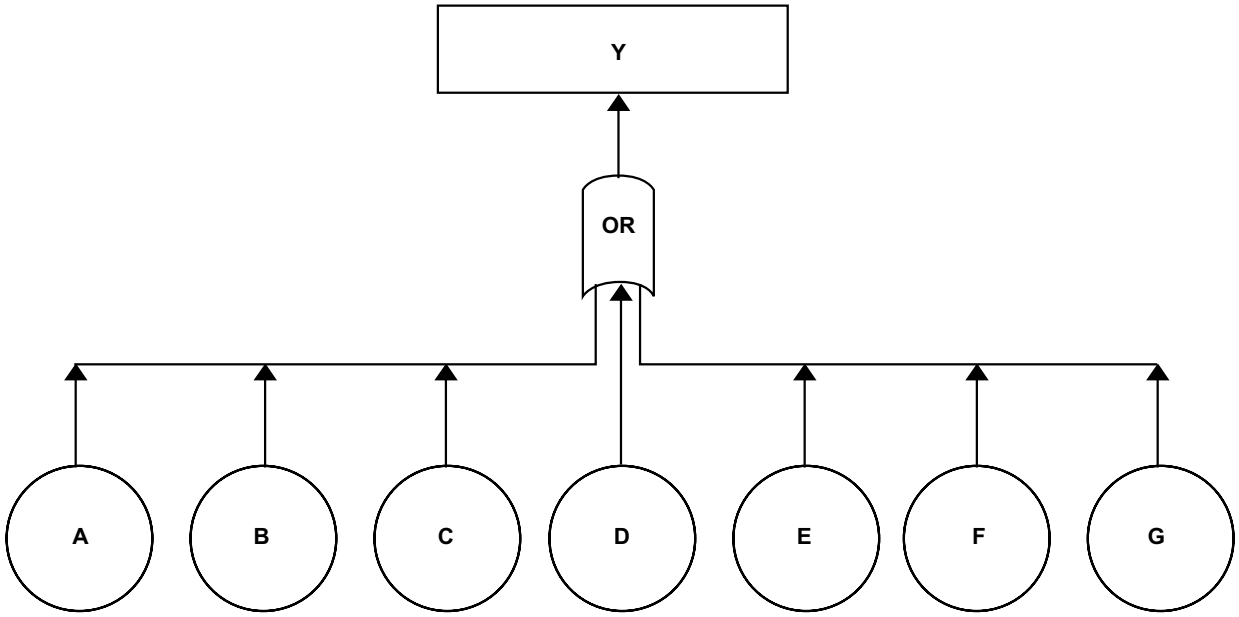

Figure 10. Fault Tree Analysis for Non-Fault Detection Segment

Basic Events

A—CPU Register fault

B—Program Counter fault

C—Interrupt handling and execution fault

D—Clock Fault

E—Invariable memory[5] fault

F—Variable memory[6] fault

G—Analog input fault

Top Events

Y—Fault in Non-fault detection segment.

5.Memory range in a processor system containing data which is not intended to vary during program execution.

6.Memory range in a processor system containing data which is intended to vary during program execution.

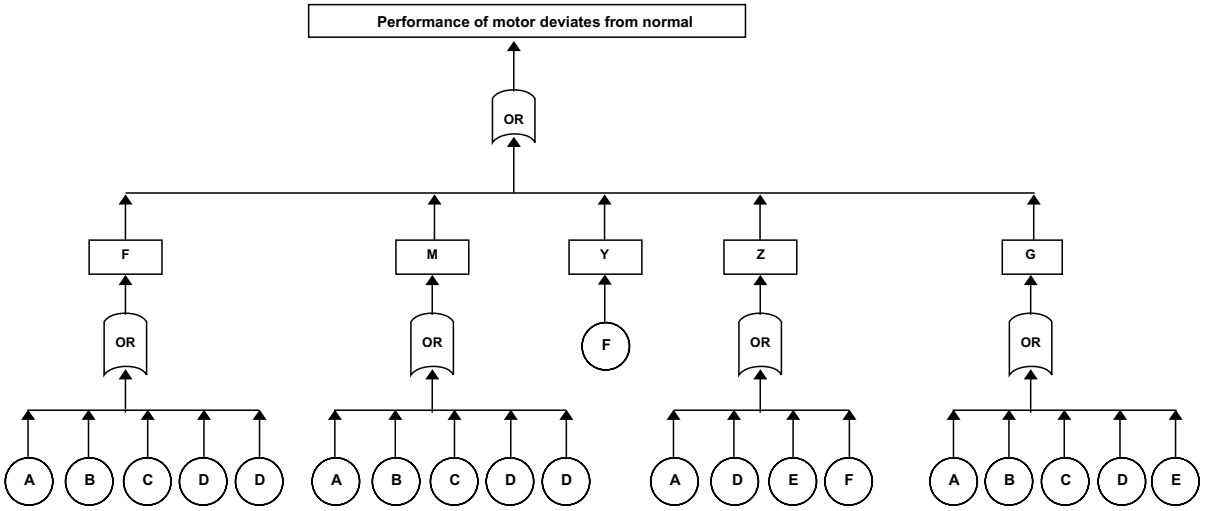

Figure 11. Detailed Fault Tree Analysis for Non-Fault Detection Segment

Basic Events

A—Register fault

B—Interrupt handling and execution fault

C—Clock Fault

D—Invariable memory fault

E—Variable memory fault

Intermediate Events

F—Timer0 module related fault

M—ADC module related fault

N—OpAmp module related fault

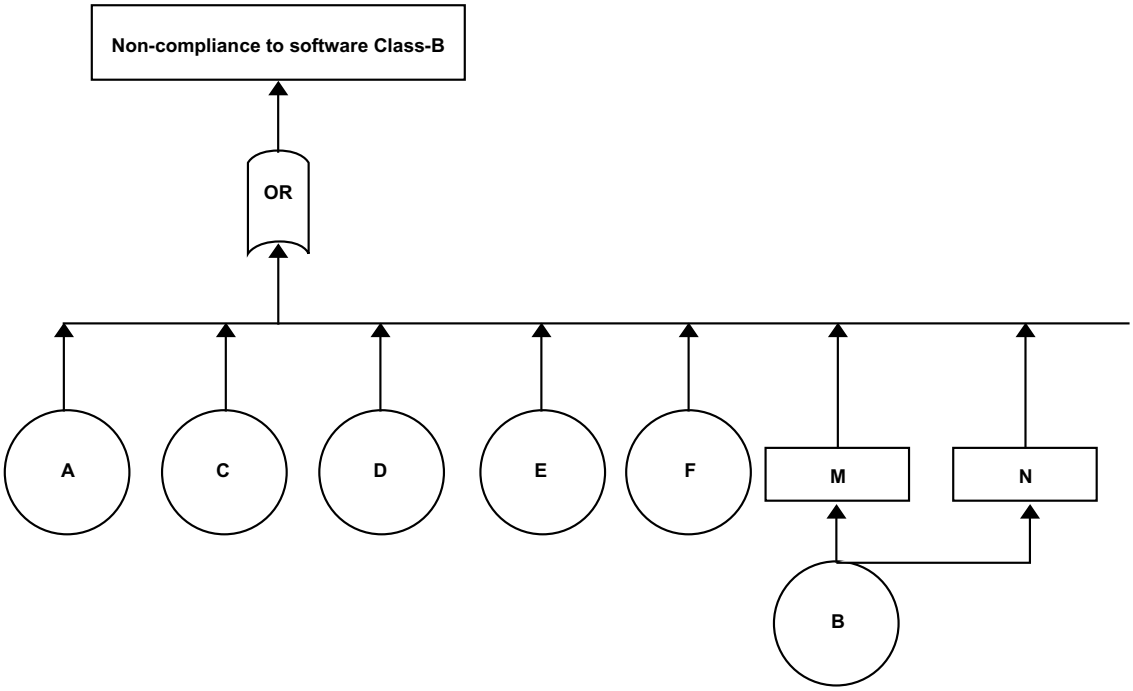

Figure 12. Detailed Fault Tree Analysis for Fault Detection Segment

Basic Events

A—CPU register faults

B—WDT module faults

C—Clock Faults

D—Invariable memory faults

E—Variable memory faults

F—Analog input faults

Intermediate Events

M—Fault to detect a fault related to interrupt handling

Appendix B—Acceptable Measures Adopted in MC Library v1.1.0 for Fault Detection

Table 4. Fault and Acceptable Measures

No	Component	Sub-component	Fault/Error	Acceptable Measures
1	CPU	Register	Stuck at	Periodic self-test using static memory test.
		Program Counter	Stuck at	Time slot monitoring.
2	Interrupt handling and execution		No interrupt or too frequent interrupt	Time slot monitoring.
3	Clock		Wrong frequency	Clock failure detection module.
4	Memory	Invariable memory	All single bit faults	Word protection with single bit redundancy.
		Variable memory	DC fault	Word protection with single bit redundancy and static memory test.
		Addressing (both variable and invariable memory)	Not valid for Z8FMC16100, with internal memory. Previous tests conducted on RAM and Flash memory would highlight any fault.	
5	Internal data path	Not valid for Z8FMC16100, with internal memory. Previous tests conducted on RAM and Flash memory would highlight any fault.		
6	External Communication	No external communication is used in MC Library v1.1.0.		
7	Input/Output Periphery	DC bus voltage	Over voltage	Plausibility check
			Under voltage	
			Over current	
8	Custom chips (ASIC, GAL)	No ASIC, GAL MC 1.1.0.		

Appendix C—Control Response for Faults in MC Library v1.1.0

Table 5. Faults and Control Response

Fault	Detection of Fault	Control Response for Fault
Struck at fault in CPU register	Static Memory test is performed on the registers to detect stuck at fault. Since, this fault can occur any time when CPU is running, the test must be done periodically to detect occurrence of faults.	Stop the motor if fault is detected.
Stuck at fault in program counter	A WDT is run and is set to Reset the CPU when it overflows. When program counter is faulty, normal program sequence is disturbed and hence WDT overflows.	Reset the CPU.
Faulty Interrupt	WDT to monitor interrupt occurrence.	Stop the motor if fault is detected.
Wrong clock frequency	Detection of wrong clock frequency is done by an on-chip hardware module that monitors the clock frequency, periodically.	Stop the motor if fault is detected.
Single bit fault in Flash memory	During startup, check the content of Flash memory and write redundant bit based on parity for each byte at a different location. During run-time, verify the parity bit for each byte.	Stop the motor if fault is detected.
DC fault in RAM	When writing data for a RAM variable, verify the content and write redundant bit based on parity at a different location. During read, verify the parity bit.	Stop the motor if fault is detected.
Stuck at fault in Flash and RAM	Carry out single redundant bit test.	Stop the motor if fault is detected.
Fault in analog input/ output and multiplexer	Periodically check if the values are within limits.	Stop the motor if fault is detected.

Appendix D—Unit Test

Unit test carried out on MC Library v1.1.0 is implemented through a unit test plan. The unit test plan broadly covers the following sections:

Test Topology

This section explains the topology of test to be carried out. A schematic displays the interconnection of ZiLOG's Z8FMC16100 board, power supply, debugger, and serial port to a PC. The topology is helpful for test engineers to understand the test environment.

Hardware

The test is conducted on ZiLOG's Z8FMC16100KIT along with the motor control application board. Motor is Linix 3-phase BLDC motor supplied with the kit. Program is downloaded through opto-isolated debugger and command for test input sent through UART interfaced to COM2.

Software

The following software required for test on MC Library v1.1.0 —ZDS II Z8 Encore! v4.10.0, HyperTerminal. The test-code is designed which includes initialization routine for UART and it accepts input based on which a particular fault is simulated.

Unit Test Case by Simulating CPU Register Fault

The code for MC Library v1.1.0 is loaded into the Flash memory of Z8FMC16100 and the motor starts functioning. Then the command is passed to the test-code through HyperTerminal, which simulates a register fault condition. The fault detection routine of MC Library v1.1.0 detects the fault conditions by performing periodic static memory tests on register and brings the controller to STOP mode. Hence, the motor stops.

Unit Test Case by Simulating Program Counter Fault

The code for MC Library v1.1.0 is loaded into the Flash memory of Z8FMC16100 and the motor starts functioning. The program counter is not accessible directly to create a fault, a program counter fault is simulated by executing an infinite loop through the command from the HyperTerminal. As per the acceptable measures for MC Library v1.1.0, the WDT overflows after the specified time and Resets the controller.

Unit Test Case by Simulating Interrupt Fault

The code for MC Library v1.1.0 is loaded into the Flash memory of Z8FMC16100 and the motor starts functioning. Interrupt fault condition is simulated by sending command from HyperTerminal, which executes DI-instruction. The fault detection routine of MC Library v1.1.0 detects the fault conditions and brings the controller to STOP mode. Hence, the motor stops.

Unit Test Case by Creating Clock Fault

The code for MC Library v1.1.0 is loaded into the Flash memory of Z8FMC16100 and the motor starts functioning. A clock failure condition is created by shorting the XIN pin to ground. The fault detection routine of MC Library v1.1.0 detects the clock fault and generates an interrupt. The WDT then takes over as the clock source and executes the interrupt service routine which brings the controller to STOP mode. Hence, the motor stops.

Unit Test Case by Simulating Flash Fault

The code for MC Library v1.1.0 is loaded into the Flash memory of Z8FMC16100 and the motor starts functioning. Then the command is passed to the test-code through HyperTerminal, which simulates a Flash fault condition at any five different memory-locations which holds the code for MC Library. The fault detection routine of MC Library v1.1.0 detects the fault condition by performing single bit redundancy test and brings the controller to STOP mode. Hence, the motor stops.

Unit Test Case by Simulating RAM Fault

The code for MC Library v1.1.0 is loaded into the Flash memory of Z8FMC16100 and the motor starts functioning. Then the command is passed to the test-code through HyperTerminal, which simulates a RAM fault condition at any memory-locations of RAM (`0x00` to `0xFF` for small memory model and `0x000` to `0x1FF` for large memory model). The fault detection routine of MC Library v1.1.0 detects the fault conditions by performing static memory test and brings the controller to STOP mode. Hence, the motor stops.

Unit Test Case by Creating Over Voltage Fault

The code for MC Library v1.1.0 is loaded into the Flash memory of Z8FMC16100 and the motor starts functioning. Then the supply voltage for DC-bus is suddenly increased beyond 25 V creating a condition of over voltage fault. The fault detection routine of MC Library v1.1.0 detects the fault conditions and brings the controller to STOP mode. Hence, the motor stops.

Unit Test Case by Creating Over Voltage Fault

The code for MC Library v1.1.0 is loaded into the Flash memory of Z8FMC16100 and the motor starts functioning. Then the supply voltage for DC-bus is suddenly reduced below 20 V creating a condition of under voltage fault. The fault detection routine of MC Library v1.1.0 detects the fault conditions and brings the controller to STOP mode. Hence the motor stops.

Unit Test Case by Creating Over Voltage Fault

The code for MC Library v1.1.0 is loaded into the flash memory of Z8FMC16100 and the motor starts functioning. Then the shaft of motor is loaded by holding it, by which the current drawn by motor increases, creating a condition of over current fault. The fault detection routine of MC Library v1.1.0 detects the fault conditions and brings the controller to STOP mode. Hence the motor stops.

The above tests were repeated for SPEED mode and TORQUE mode of MC library and for small memory model and large memory model.

The tests were retried on four different trials and results are recorded in unit test report. The table below mentions the plan for test case for clock fault, as an example for format:

Test Case ID	UT_Clock_Fault_025
Description	This is a negative test case for detecting a Wrong Frequency fault in clock in SPEED mode and Small Memory model.
Test topology	Test set up is described in test topology (section -3.1)
Precondition	None
Inputs	No input parameters for this API.
Procedure	Once the motor starts running smoothly, the clock fault condition is created by shorting the XIN pin to GND.
Outputs	Motor starts.
Remarks if any	

The table below summarizes the tests carried out as a part of unit test plan on MC Library v1.1.0 and the way each test is carried out to detect the faults as described in the fault tree analysis (Appendix A).

Unit Test Case for Fault	Relation to Fault Tree Analysis (Appendix A)	Relation to Acceptable Measure (Appendix B)	Relation to Control Response	Unit Test Case ID
Register fault	Basic event — 'A' of Figure 11 of Appendix A	Sl No 1 of Table 7 of Appendix B	Stop the motor if fault detected	UT_Register_Fault_005, UT_Register_Fault_006, UT_Register_Fault_007, UT_Register_Fault_008
PC fault	Basic event — 'B' of Figure 11 of Appendix A	Sl No 1 of Table 7 of Appendix B	Reset CPU	UT_PC_Fault_009, UT_PC_Fault_010, UT_PC_Fault_011, UT_PC_Fault_012
Interrupt fault	Basic event — 'C' of Figure 11 of Appendix A	Sl No 2 of Table 7 of Appendix B	Stop the motor if fault detected	UT_Interrupt_Fault_013, UT_Interrupt_Fault_017, UT_Interrupt_Fault_021, UT_Interrupt_Fault_014, UT_Interrupt_Fault_018, UT_Interrupt_Fault_022, UT_Interrupt_Fault_015, UT_Interrupt_Fault_019, UT_Interrupt_Fault_023, UT_Interrupt_Fault_016, UT_Interrupt_Fault_020, UT_Interrupt_Fault_024
Clock fault	Basic event — 'D' of Figure 11 of Appendix A	Sl No 3 of Table 7 of Appendix B	Stop the motor if fault detected	UT_Clock_Fault_025, UT_Clock_Fault_026, UT_Clock_Fault_027, UT_Clock_Fault_028
Flash fault	Basic event — 'E' of Figure 11 of Appendix A	Sl No 4 of Table 7 of Appendix B	Stop the motor if fault detected	UT_Flash_Fault_029, UT_Flash_Fault_030, UT_Flash_Fault_031, UT_Flash_Fault_032
RAM fault	Basic event — 'F' of Figure 11 of Appendix A	Sl No 4 of Table 7 of Appendix B	Stop the motor if fault detected	UT_RAM_Fault_033, UT_RAM_Fault_037, UT_RAM_Fault_034, UT_RAM_Fault_038, UT_RAM_Fault_035, UT_RAM_Fault_039, UT_RAM_Fault_036, UT_RAM_Fault_040

Over voltage fault	Basic event — 'G' of Figure 11 of Appendix A	Sl No 7 of Table 7 of Appendix B	Stop the motor if fault detected	UT_Analog_Fault_044, UT_Analog_Fault_041
Under voltage fault	Basic event — 'G' of Figure 11 of Appendix A	Sl No 7 of Table 7 of Appendix B	Stop the motor if fault detected	UT_Analog_Fault_045, UT_Analog_Fault_042
Over current fault	Basic event — 'G' of Figure 11 of Appendix A	Sl No 7 of Table 7 of Appendix B	Stop the motor if fault detected	UT_Analog_Fault_046, UT_Analog_Fault_043

Appendix E—Configuration Management of MC Library v1.1.0

The following steps describe the configuration management of MC Library v1.1.0:

1. Tool used for version control—Star Team v7.0.133.
2. Requirement Traceability—All requirements from software design specification up to development and testing of code are maintained in 'Requirement traceability matrix'.
3. Phases of software development—ZiLOG has a detailed process map followed during the software development phase. The development process has 5 phases from start to end. Each phase has a set of well-defined inputs, tasks, deliverables, and responsibility. Figure 13 illustrates the phase wise folders recorded in Star Team.

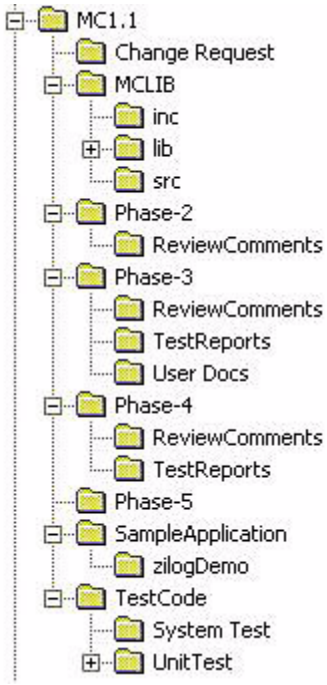

Figure 13. Process Map

4. Change management—Any change required to be made to the software code has to go through the 'Change Request' (CR) process, available in Star Team. Figure 14 illustrates the sample change request.

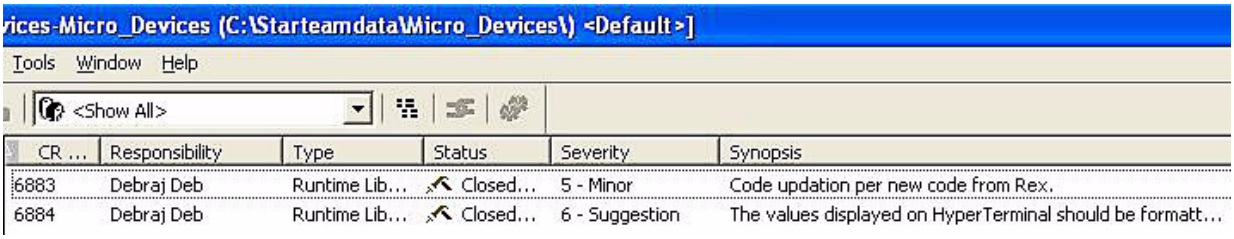

Figure 14. Sample Change Request

The CR process involves the following steps:

- Enter the CR on Star Team.
- Assign CR to person who will incorporate the change.
- Open status—Currently the change is not implemented and hence CR is OPEN.
- Fixed status—Change is implemented.
- Fixed Verified—Confirmation
- Closed (fixed)—CR closed.
- Depending on the type of request, the status can be ‘Cannot Reproduce, As Designed, Documented, Is Duplicate or Differed’ before the CR enters the Closed status.

CR also records the test case ID and the version number for implemented change, and through the CR process a complete control and monitoring of change management is done.

5. Configuration identification scheme: ZiLOG core quality process specifies the naming standard for each ‘base-line’, once the code is successfully tested and is done during the exit of phase-4. The same process applies for MC Library v1.1.0. This ‘base-line’ scheme ensures that the version of software is frozen in the system and is recalled any time in future, based on the ‘base-line’. Figure 15 illustrates the ‘base-line’ for MC Library v1.1.0.

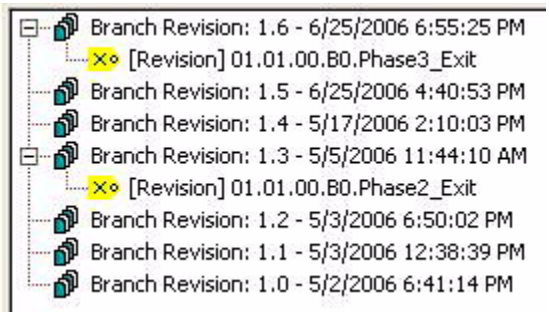

Figure 15. Base-line for MC Library v1.1.0

6. Control receipt, storage, handling, and release of configurable items listed below are stored in Star Team:
   - Project plan.
   - Software design specification.
   - Software functional specification.
   - Requirement traceability matrix.
   - Unit test plan.
   - System test plan.
   - Unit test report.
   - System test report.
   - Phase exit checklist.
   - Release checklist.
   - Characterization report.
   - User documents—User Manual, Reference Manual, Quick Start Guide, and Product Brief.

This publication is subject to replacement by a later edition. To determine whether a later edition exists, or to request copies of publications, contact:

ZiLOG Worldwide Customer Support Center
532 Race Street
San Jose, CA 95126
USA
Telephone: 408.558.8500
Fax: 408.558.8300
www.zilog.com

Information Integrity

The information contained within this document has been verified according to the general principles of electrical and mechanical engineering. Any applicable source code illustrated in the document was either written by an authorized ZiLOG employee or licensed consultant. Permission to use these codes in any form, besides the intended application, must be approved through a license agreement between both parties. ZiLOG will not be responsible for any code(s) used beyond the intended application. Contact the local ZiLOG Sales Office to obtain necessary license agreements.

Document Disclaimer