Université du Québec
École de technologie supérieure

RAPPORT TECHNIQUE
PRÉSENTÉ À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
DANS LE CADRE DU COURS MGL 804 REALISATION ET MAINTENANCE DE LOGICIEL

KEY PROCESS AREAS FOR SYSTEMS ENGINEERING DESGIN AND TEST THROUGH V-CYCLE

MOUSTAFA MOUSTAFA
MOUM09018308

DÉPARTEMENT DE GÉNIE LOGICIEL ET DES TI

Professeur superviseur
ALAIN APRIL

MONTRÉAL, 17 APRIL 2012
WINTER 2012

SOFTWARE EVOLUTION AND CORRECTION (DESGIN AND TEST THROUGH V-CYCLE)

MOUSTAFA MOUSTAFA
MOUM09018308

RÉSUMÉ

The document will involve a real application (Light system control software) detailed design and unit testing as trial for addressing part of the system engineering (electronics hardware and software integration) for one key process area (software evolution and correction) through support to evolution engineering to be able to be achieved in CMMI.

Table of content

1 Introduction

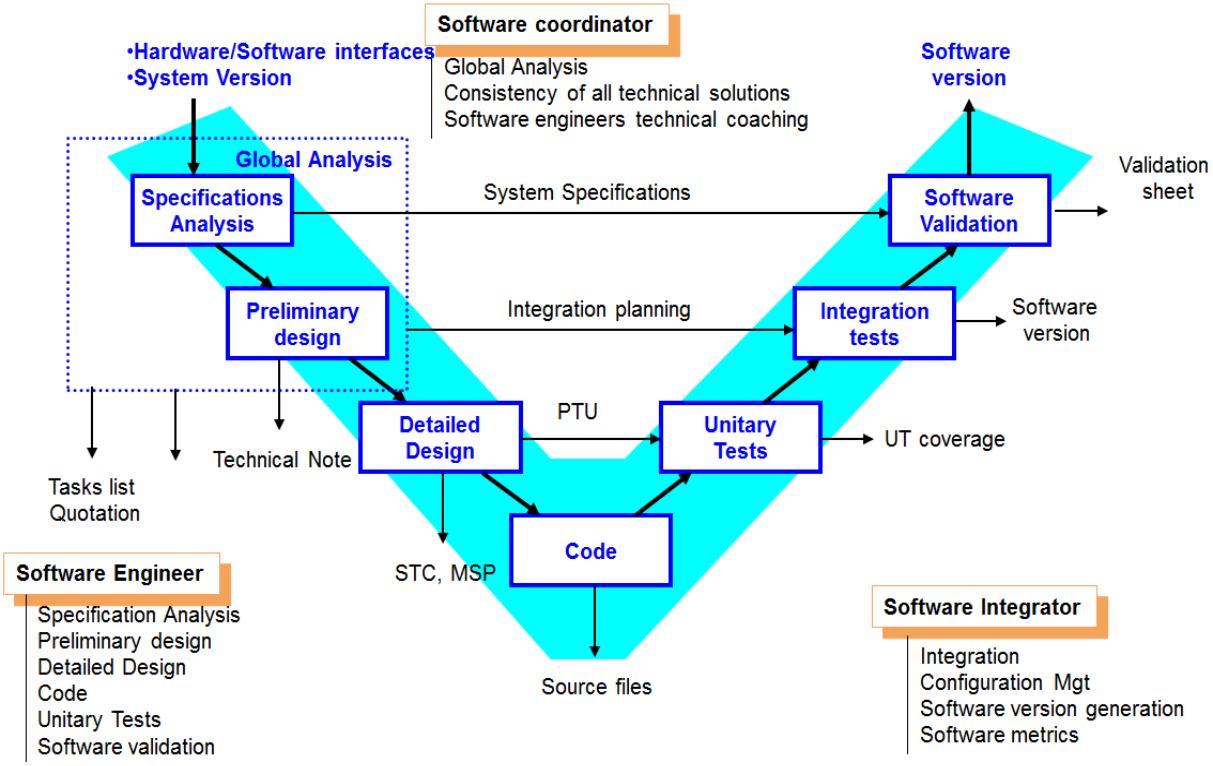
A definition of system engineering should be settled first to be able to achieve it in CMMI. Simply, System engineering can be defined as gathering software, electronics hardware, mechanical hardware, and controls together in one package. So initially, it seems very difficult to address System engineering in CMMI as it is very complex. Complexity here means gathering all the standards (software standard, electronics hardware standard, mechanical hardware standard, and controls standard) together in one standard or at least finding the area of intersection between these standards to initiate system engineering in CMMI.

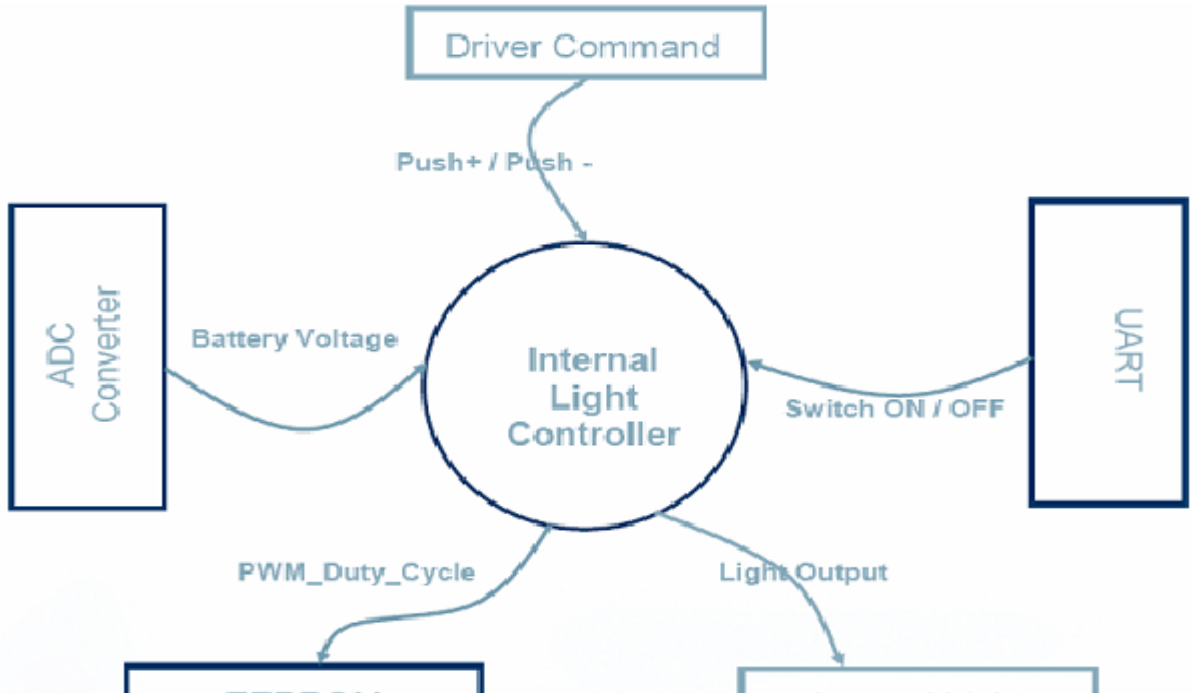
2 Methodologies

Two existing methodologies to address initiation of system engineering in CMMI. First, a very high level approach by means of general overview without details for all the key process areas to be applied in all components (software, electronics hardware, mechanical hardware, and controls) of system engineering in CMMI. Second, a very low level approach which is picking up only two components from system engineering (e.g. software, and electronics hardware) for integration to address only part of one key process area (e.g. evolution engineering) with deep details in CMMI. Unfortunately, the choice will be to the second approach as the first one is very complex even if it is general overview.

3 Case Study

The case study will cover the integration of software and electronics hardware for light system control to cover the detailed design and unit testing through V-cycle to be addressed as part of system engineering in CMMI. As shown below two figures that shows overview of the system with V-cycle implementation.





4 Detailed Design

4.1 Constraints on Initialization

- Motor will be in NORMAL state.
- Gateway command will be ON.
- Interrupts will be enabled.
- Scheduled tasks will be activated.
- Watchdog will be activated.

4.2 Constraints on Inputs

N°	Input	Period / Frequency	Response Time	Jitter	Initial Requirement
Inp1	PCB_Temp		72ms	+4ms, -4ms	REQ_1, REQ_2
Inp2	Battery voltage		24ms	+4ms, -4ms	REQ_3, REQ_4
Inp3	Gateway		120,000ms		REQ_13, REQ_14
Inp4	Short circuit		0.045ms		REQ_17, REQ_18

4.3 Constraints on Outputs

N°	Output	Period / Frequency	Event	Response Time	Jitter	Initial Requirement
Out1	Battery Fault		Inp2 (Fault happened)	24ms	+4ms, -4ms	REQ_3, REQ_10
Out2	Battery Clear		Inp2 (Fault disappears)	24ms	+4ms, -4ms	REQ_4, REQ_10
Out3	Temperature Fault		Inp1 (Fault happened)	72ms	+4ms, -4ms	REQ_1, REQ_10
Out4	Temperature Clear		Inp1 (Fault disappears)	72ms	+4ms, -4ms	REQ_2, REQ_10
Out6	GatewayOutput		Inp3 (Gateway Voltage > 10v)	120,000ms		REQ_15
Out7	Diagnostic display battery fault		Inp2 (Fault happened)	24ms	+4ms, -4ms	REQ_7

Out9	Diagnostic display temp fault		Inp1 (Fault happened)	72ms	+4ms, -4ms	REQ_7
Out10	Diagnostic display temp clear		Inp1 (Fault disappears)	72ms	+4ms, -4ms	REQ_8
Out11	Motor command fault		Inp1, Inp2			REQ_20,
Out12	Short circuit output	10,000	Inp4	0.045ms		REQ_17, REQ_18
Out13	Watchdog refresh	50ms		30ms	±20ms	REQ_11
Out14	Motor init		initialization			REQ_19

5 Real Time Analysis

Four tasks will be scheduled; vidBatVolAcqResponse, vidTempAcqResponse, and vidGateAcqResponse, and vidWatchdog. Every time “T” ms, these four tasks will be called and executed.

5.1 Processes Identification

Response time = ((estimated duration/ ”T”) * 100)% of “T” ms

So, e.g. Response time for Inp1 (PCB Temp) = ((0.15ms/T)*100) = (15/T)% of “T” ms

Jitter = (((total jitter / (total periodicity/T))/T) * 100))% of “T” ms

So, e.g. Jitter for Inp1 (PCB Temp) = ((4ms/(72ms/Tms))T)*100 = 5.55% of “T” ms

Constraints are allocated to processes:

Constraint	Basic operation	Process	Period / Frequency	Response Time	Jitter
Out1	Battery Fault	vidBatVolAcqResponse	“T”ms	(20/T)% of T ms	±16.6% of ”T”ms
Out2	Battery Clear	vidBatVolAcqResponse	“T”ms	(20/T)% of T ms	±16.6% of ”T”ms
Out3	Temperature Fault	vidTempAcqResponse	“T”ms	(15/T)% of T ms	±5.55% of ”T”ms
Out4	Temperature Clear	vidTempAcqResponse	“T”ms	(15/T)% of T ms	±5.55% of ”T”ms
Out6	GatewayOutput	vidGateAcqResponse	“T”ms	(20/T)% of T ms	
Out7	Diagnostic display battery fault	vidBatVolAcqResponse	“T”ms	(120/T)% of T ms	±16.6% of ”T”ms
Out8	Diagnostic display battery clear	vidBatVolAcqResponse	“T”ms	(120/T)% of T ms	±16.6% of ”T”ms
Out9	Diagnostic display temp fault	vidTempAcqResponse	“T”ms	(115/T)% of T ms	±5.55% of ”T”ms
Out10	Diagnostic display temp clear	vidTempAcqResponse	“T”ms	(115/T)% of T ms	±5.55% of ”T”ms
Out11	Motor command fault	vidTempAcqResponse or vidBatVolAcqResponse	“T”ms	(15/T)% of T ms or (20/T)% of T ms	±5.55% of ”T”ms or ±16.6% of ”T”ms
Out12	Short circuit output	vidIsrShortCircuit	10,000ms	0.045	

Out14	Motor init	vidInit			
Inp1	PCB_Temp	vidTempAcqResponse	“T”ms	(15/T)% of T ms	±5.55% of ”T”ms
Inp2	Battery voltage	vidBatVolAcqResponse	“T”ms	(20/T)% of T ms	±16.6% of ”T”ms
Inp3	Gateway	vidGateAcqResponse	“T”ms	(20/T)% of T ms	
Inp4	Short circuit	vidIsrShortCircuit		0.045ms	

5.2 Timing Bases Identification

Main constraints on processes are identified to define a timing basis for each process:

Process	Description	Constraints	Period	Response Time	Jitter
vidTempAcqResponse	Taking 3 consecutive acquisition for PCB temperature, computes their mean, defines if fault happened or not and outputs the result to writeEEPROM and diagnosticDisplay	Inp1 Out3 Out4 Out9 Out10 Out11	“T“ms	(115/T)% of T ms	±5.55% of ”T”ms
vidBatVolAcqResponse	Taking 4 consecutive acquisition for battery voltage, computes their mean, defines if fault happened or not and outputs the result to writeEEPROM and diagnosticDisplay	Inp2 Out1 Out2 Out7 Out8 Out11	“T“ms	(120/T)% of T ms	±16.6% of ”T”ms
vidGateAcqResponse	Taking 4 consecutive acquisition for gateway voltage, computes their mean, defines if voltage > 10v, so gateway is switched on and outputs signal of 40µs	Inp3 Out6	“T“ms	(20/T)% of T ms	
vidWatchdog	Refreshment of watchdog	Out13	50ms	30ms	±20ms

vidInit	Initialization routine for the component	Out14			
vidIsrShortCircuit	Interrupt service routine for short circuit	Inp4 Out12	10,000ms	0.045	

5.3 Shared Resources

Hardware Resources: No hardware resources shared.

Software Resources

Sharing for tenuGateCmd (which represents the gateway command ON OFF)will be between vidInit to initialize it and vidGatAcqResponse that uses its value and updates it by the new command, here, there will be no need for protection because vidInit will be called first before the task vidGatAcqResponse. Also the same case for u8MainFlag that will be initialized in vidInit by 1 to enable the tasks to be scheduled, and then updating it will be after reading its value in tasks and writing in it in vidInit, so no need for protection.

In case of u8ShortCircuitFlag that used to know whether there is a short circuit or not, it will be shared between vidIsrShortCircuit that updates it by 1 indicating that there is a short circuit and vidIsrCyclicTimer that updates it by 0 after 10 seconds to indicate that short circuit is finished. The problem at the moment after the vidIsrCyclicTimer updating u8ShortCircuitFlag by 0 indicating that 10 seconds finish, at that moment, there is a possibility to have vidIsrShortCiruit updating u8ShortCircuitFlag by 1 indicating that there is another new short circuit, in this case, the counter for seconds will counts forever in vidIsrCyclicTimer because it was adjusted at 10 seconds only and not 20 seconds, so we need a protection for that flag.

Services	tenuGateCmd	U8ShortCircuitFlag	U8MainFlag
vidTempAcqResponse	-	-	R
vidBatVolAcqResponse	-	-	R
vidGateAcqResponse	r/w	-	R
vidIsrSimpleTimer	-	-	-
vidIsrCyclicTimer	-	r/w	W
vidInit	W	-	W
vidIsrShortCircuit	-	W	-

Software Resource	***Protection Status***
tenuGateCmd	Not needed, writing will be before enabling interrupts in vidInit()
U8ShortCircuitFlag	Needed
U8MainFlag	Not needed, writing will be before enabling interrupts in vidInit() , also reading will be after

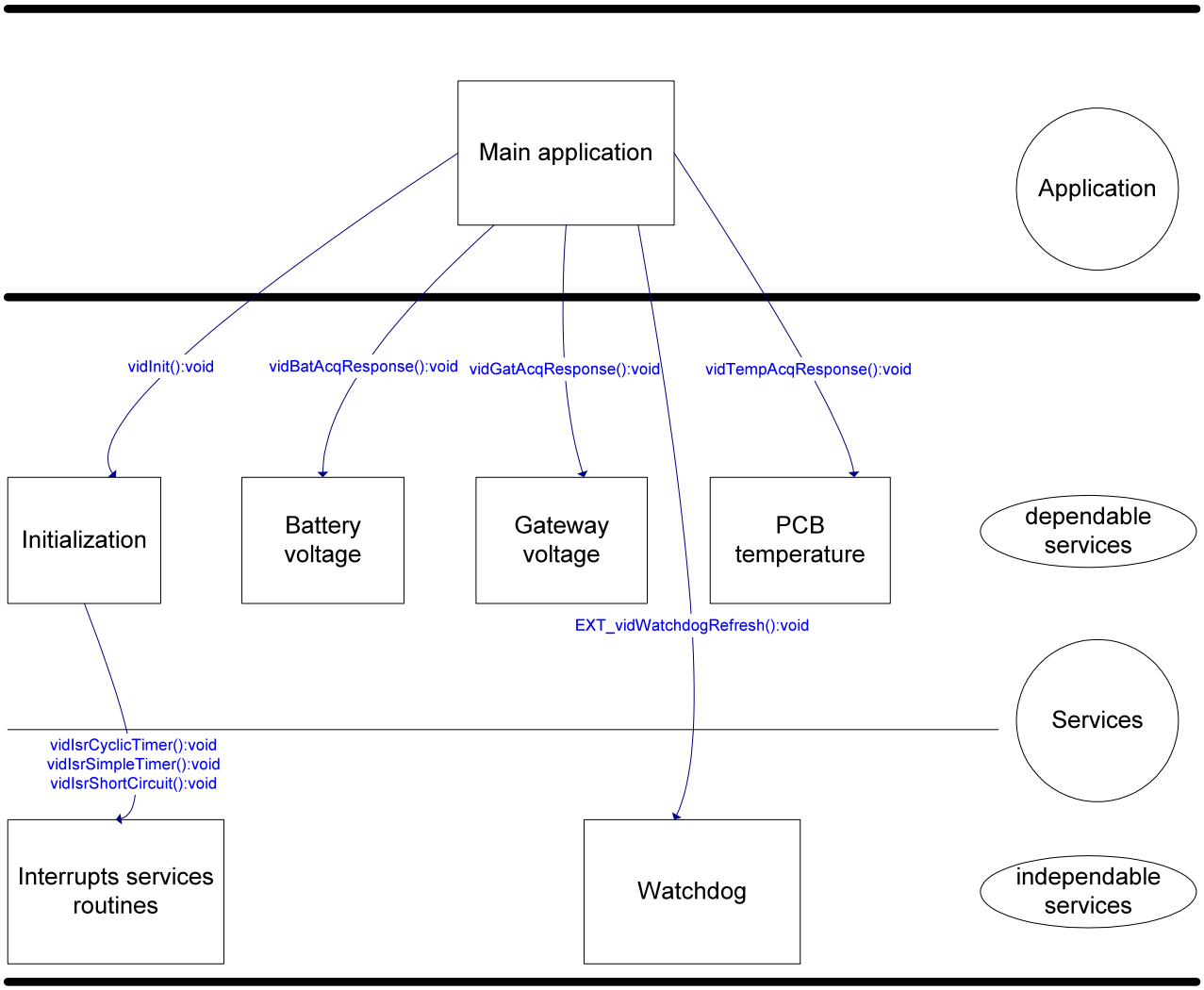
Also we will have reentrancy in stubs, The figure below shows that *EXT_vidWriteInEEPROM, and* EXT_vidCommandMotor can be called by many tasks from different context switching at the same time.

Services	***EXT_vid WriteInEEPROM***	**EXT_vid CommandMotor**
vidTempAcqResponse	√	√
vidBatVolAcqResponse	√	√
vidGateAcqResponse	-	-
vidIsrShortCircuit	-	√
vidInit	-	√

Software Resource	***Description***	***Protection Status***
EXT_vidWriteInEEPROM	Writing in EEPROM	Not needed, because it will be called by many processes but same context
EXT_vidCommandMotor	Command motor with DEGRADED or STOP or NORMAL	Not needed, because it will be called by many processes but same context

6 Architectural Design

The figure below illustrates the software architecture:



Software context consists of two main layers:

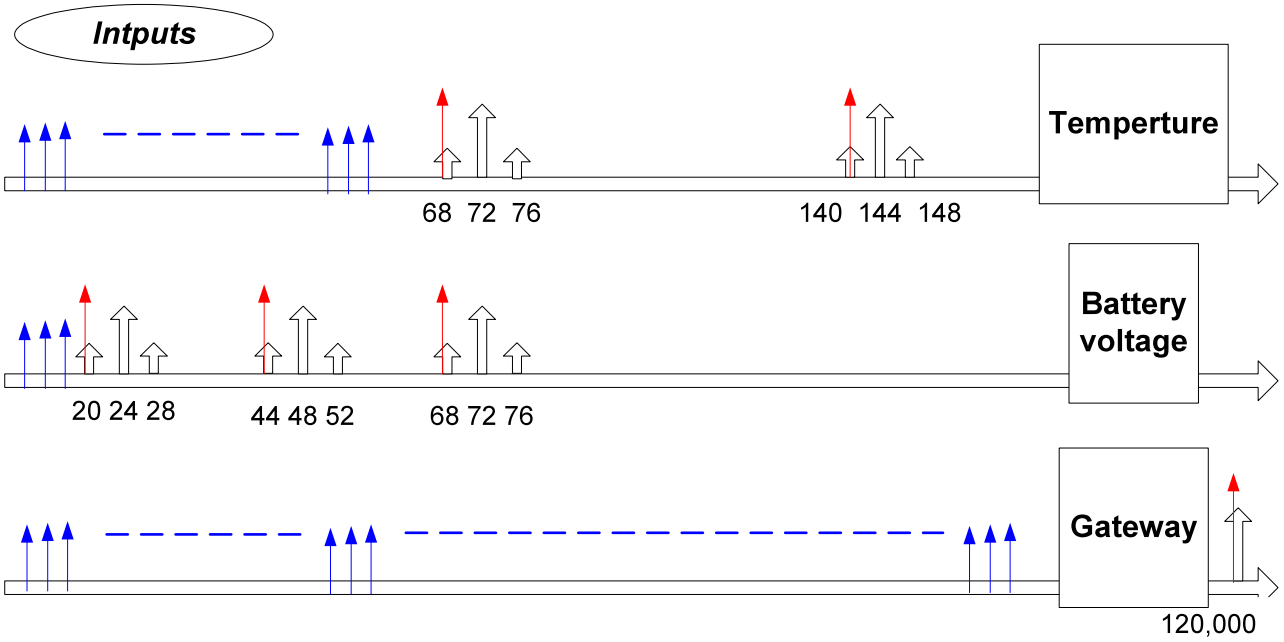
- Services layer that contains two parts:
  - First part consists of initialization, and three main tasks for temperature, battery voltage, and Gateway.
  - Second part consists of interrupts services routines, and a task for watchdog refreshment.
- Application layer that contains main application that calls the five main tasks.
-

6.1 Real Time Architecture

Note that all the numbers below in milliseconds.

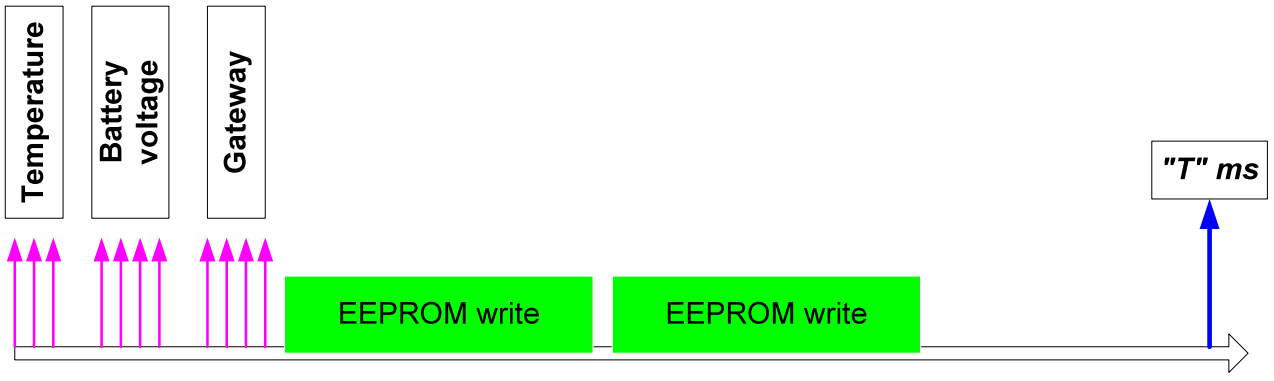
The figure below illustrates the timeline for the inputs:

The red arrow represents the response time for that input. The blue arrow represents the cyclic time “T” that will be taken for acquisition.



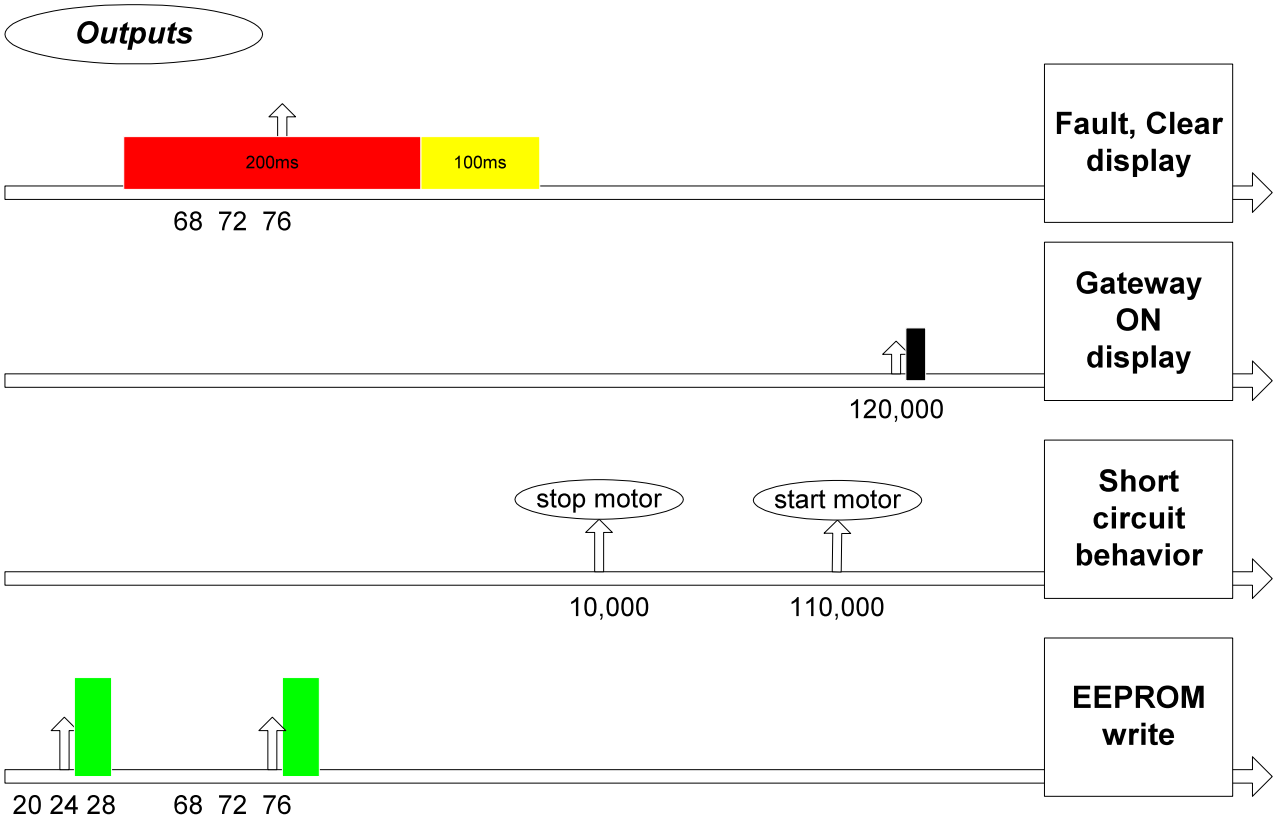
The figure below illustrates (blue arrow) the details for cyclic time “T” that used in acquisition and writing in EEPROM:

- the pink arrow represents an acquisition.
- three consecutive readings for PCB temperature that will take 150µs.
- four consecutive readings for battery voltage that will take 200µs.
- four consecutive readings for gateway voltage that will take 200µs.
- finally, in worst case that we have two faults (battery voltage, PCB temperture), so two consecutive writing in EEPROM that will take 2ms.



The figure below illustrates the timeline for outputs scenario as an example:

Consider that we have fault at 24ms, so firstly write this fault in EEPROM for 1ms (the green rectangle represents that writting), then output signal on the diagnostic display of 200ms±10ms(the red rectangle represents that signal). After that, at 76ms we have fault cleared, so firstly write this clear in EEPROM for 1ms and then waits until 200ms±10ms finished, then output signal on the diagnostic display of 100ms±10ms(the yellow rectangle represents that signal). After that, at 10,000ms, short circuit is detected, so we will stop motor before 45µs, and after 10 seconds from short circuit detection (at 110,000ms), we will start motor again. Also consider that we have gateway high voltage for 2 minutes at 120,000ms, so will output signal on the gateway of 80µs with duty cycle 50% (the black rectangle represents that signal).



6.1.1 Interrupts

Processes allocated to interrupt vectors:

IT	Period	Process	Maximum Delay	Response Time	Estimated Duration	Estimated CPU Load	IT Level
Short circuit		vidIsrShortCircuit		45µs	10µs	0.25%	1
Simple timer		vidIsrSimpleTimer	45µs		5µs	0.125%	2
Cyclic timer	4ms (will be discussed in section 3.3)	vidIsrCyclicTimer	60µs		2µs	0.05%	3
					Estimated CPU load	0.675%	

6.1.2 Cyclic Tasks

Processes allocated to cyclic tasks:

Task	Period	Process	Response Time	Estimated Duration	Estimated CPU Load	IT Level

	in section 3.3)					
PCB temperature task	4ms (will be discussed in section 3.3)	vidTempAcqResponse		1215µs	30.375%	3
Gateway task	4ms (will be discussed in section 3.3)	vidGateAcqResponse		255µs	6.375%	3
				Estimated CPU load	68.525%	

6.1.3 Estimated Workload

Estimated total CPU load:

- Worst case = 68.525% + 0.675% = 69.2% , in which a short circuit can happen and there are faults for battery voltage, PCB temperature, and gateway at every response time for them, these faults need to be written in eeprom that will takes 1ms for every fault.
- Best case = 18.35% + 0.425% = 18.775% , in which no short circuit and faults at all.

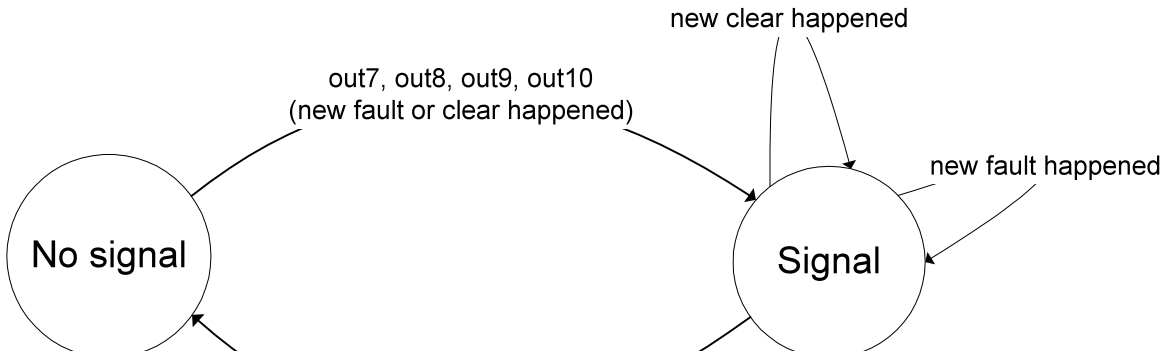
6.1.4 Protection Mechanisms

Protection mechanism that will be used for protecting short circuit flag (u8ShortCircuitFlag) is critical section; simply before updating u8ShortCircuitFlag disable interrupts and after updating the value enable interrupts. This will prevent short circuit to get in an infinite loop as discussed in section 2.3.2

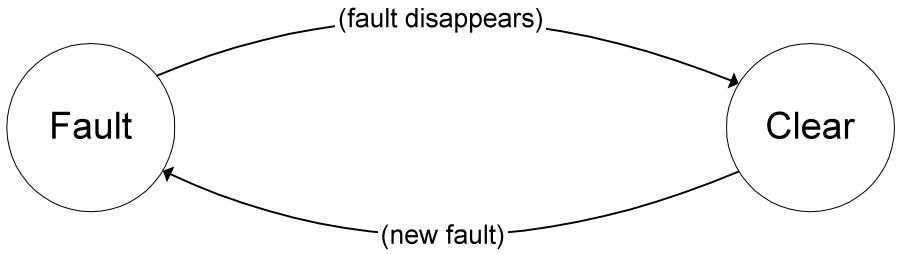
6.2 Modes Description

Diagnostic display modes:

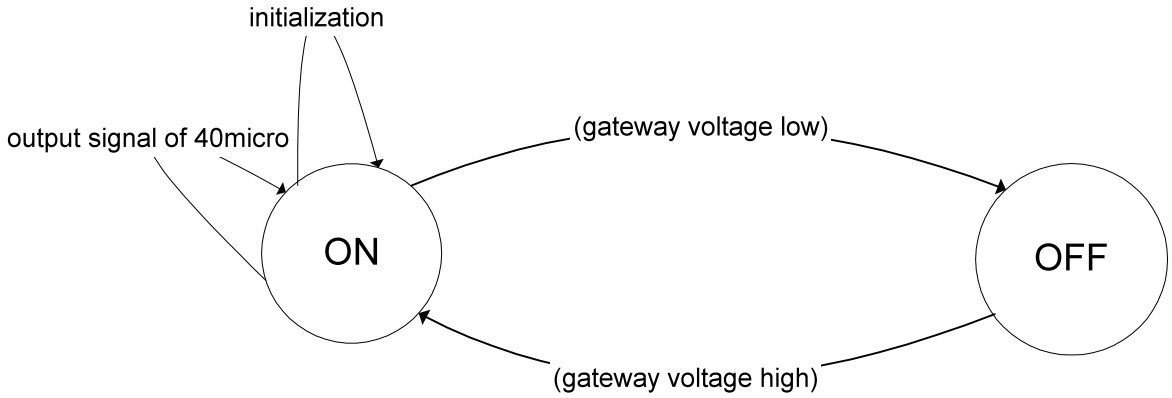
First, if the display has no signal and fault happened, so we will move from no signal state to signal state (ouput 200ms±10ms). In signal state, if fault is cleared, then we still in signal state (ouput 100ms±10ms). In signal state, if no change in fault, so move to no signal state in which there is no signal in display.



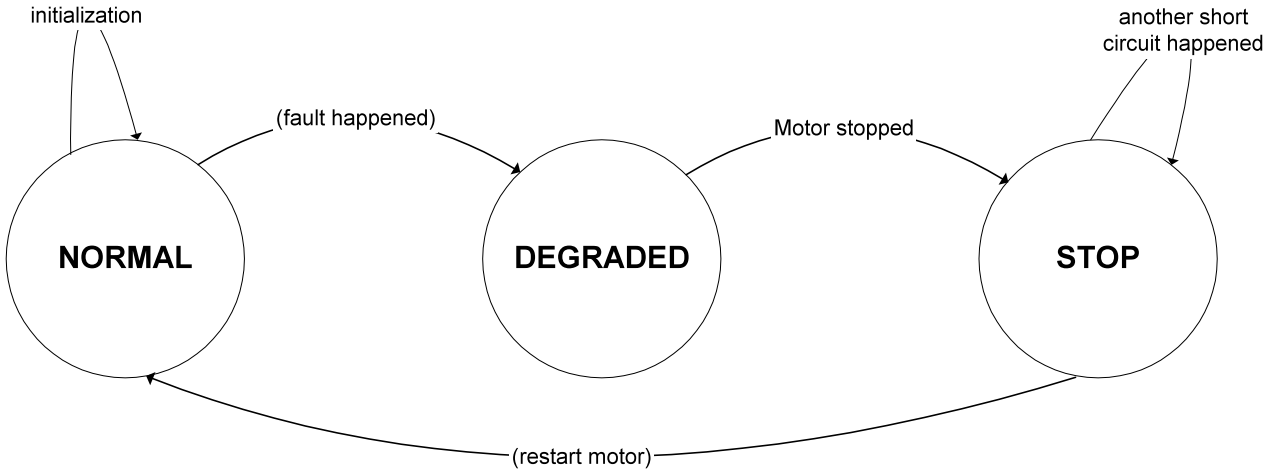
PCB temperature and battery voltage modes:



Gateway modes:



Motor modes:



6.3 Design Alternatives and Justification

The first method is that we have two resources for timing (simple timer and cyclic timer), these resources are going to control the following different timings:

- 200ms±10ms or 100ms±10ms output signal on the diagnostic display.
- 80µs with 50% duty cycle output signal on the gateway.
- 10 seconds waiting until short circuit finished.
- 10ms to 50 ms as a watchdog window.
- 72ms±4ms, 24ms±4ms, and 2 minutes as acquisitions for battery voltage, PCB temperature, and gateway voltage.

So, for better performance, we will put the cyclic timings (timings that has a periodicity) in the cyclic timer. For memory consumption (decreasing the use of flags), we will use cyclic timer for timings of low resolution (timings of millisecond and seconds), and simple timer for timings of high resolution (timings of microsecond).

Therefore, we will have the following:

- Cyclic timer used for:
    1- 200ms±10ms or 100ms±10ms output signal on the diagnostic display.
    2- 10 seconds waiting until short circuit finished.
    3- 10ms to 50 ms as a watchdog window.
    4- 72ms±4ms, 24ms±4ms, and 2 minutes as acquisitions for battery voltage, PCB temperature, and gateway voltage.
- Simple timer used for:
    1- 80µs with 50% duty cycle output signal on the gateway.

The second method is to choose a cyclic period that will fit 24ms±4ms, 72ms±4ms, and 120,000ms±4ms with high accuracy and low cpu load.

Design choices:

1- Cyclic period “T” = 3ms, in this case, we will have high accuracy but high cpu load ≈ 93%.

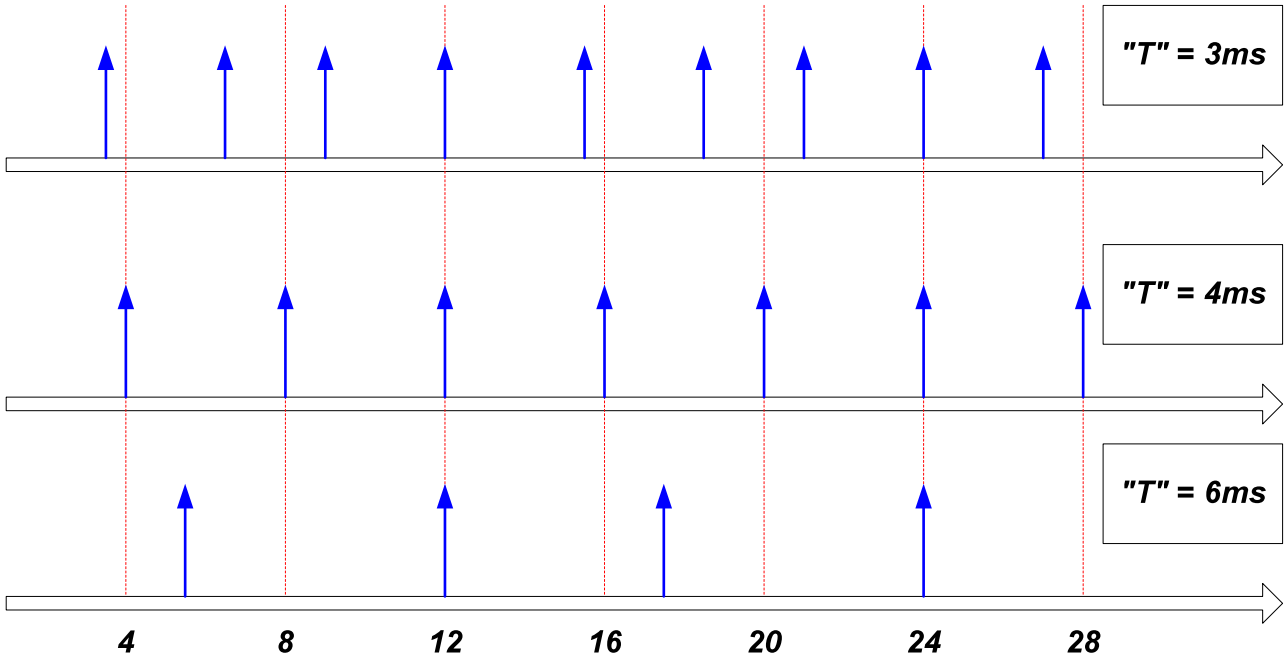
2- Cyclic period “T” = 6ms, in this case, we will have low cpu load but also low accuracy.

3- Cyclic period “T” = 4ms, in this case, we will have high accuracy and medium cpu load = 69.2%.

The figure below illustrates the three methods:

- When “T” = 3ms, we will have high accuracy to detect faults and clears very early of samples at 21ms, 24ms, and 27ms (many chances to detect the fault, if problem happened and fault not detected at 21ms, so it can be detected at 24ms, and so on) and high cpu load which is computed as cyclic tasks/cyclic period (2.5ms/3ms) ≈ 93%.
- When “T” = 4ms, we will have high accuracy to detect faults and clears very early of samples at 20ms, 24ms, and 28ms (many chances to detect the fault, if problem happened and fault not detected at 20ms, so it can be detected at 24ms, and so on) and medium cpu load which is computed as cyclic tasks/cyclic period (2.5ms/4ms) ≈ 69.2%.
- When “T” = 6ms, we will have low accuracy to detect faults and clears lately of one sample only at 24ms (one chance only, if problem happened, fault can’t be detected, which is invalid case) and low cpu load which is computed as cyclic tasks/cyclic period (2.5ms/6ms) ≈ 45%.

The same thing for 72ms and 120,000ms as 24ms.



Notice that for accuracy, we will have 3 samples for 3ms, 3 samples for 4ms and 1 sample for 6ms.

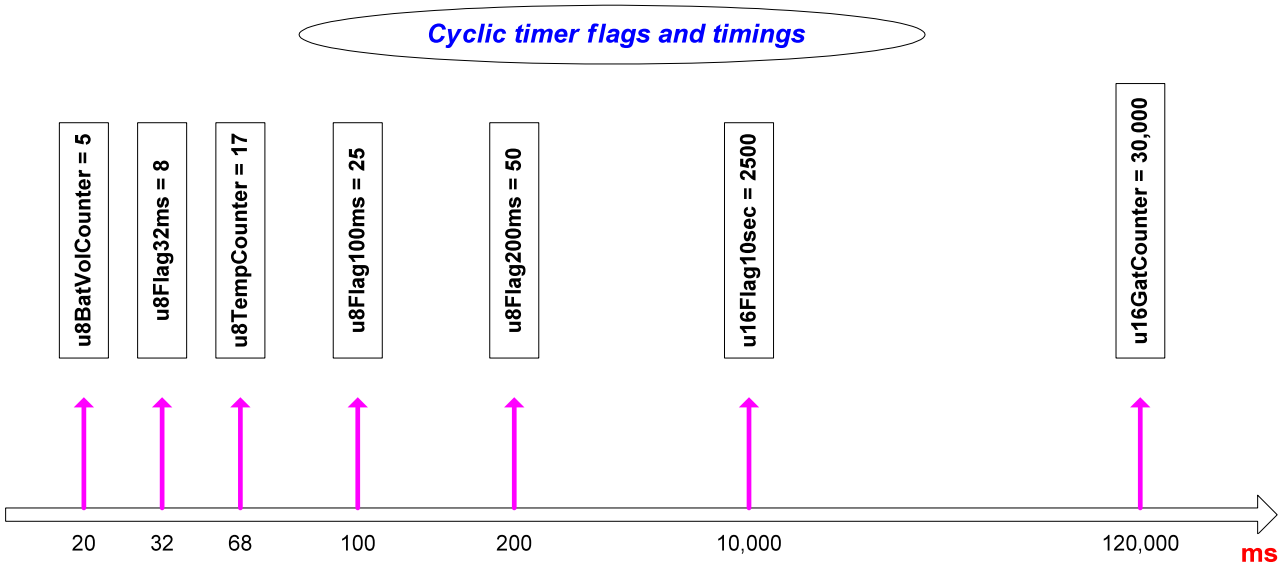
So, from the previous discussion, 3ms can't be used because of high cpu load, and 6ms can't be used because of invalid case, so we will choose third solution which is a compromise between the first two solutions in accuracy and cpu load which is "T" = 4ms.

Therefore, for the previous two methods, we will have the following timings:

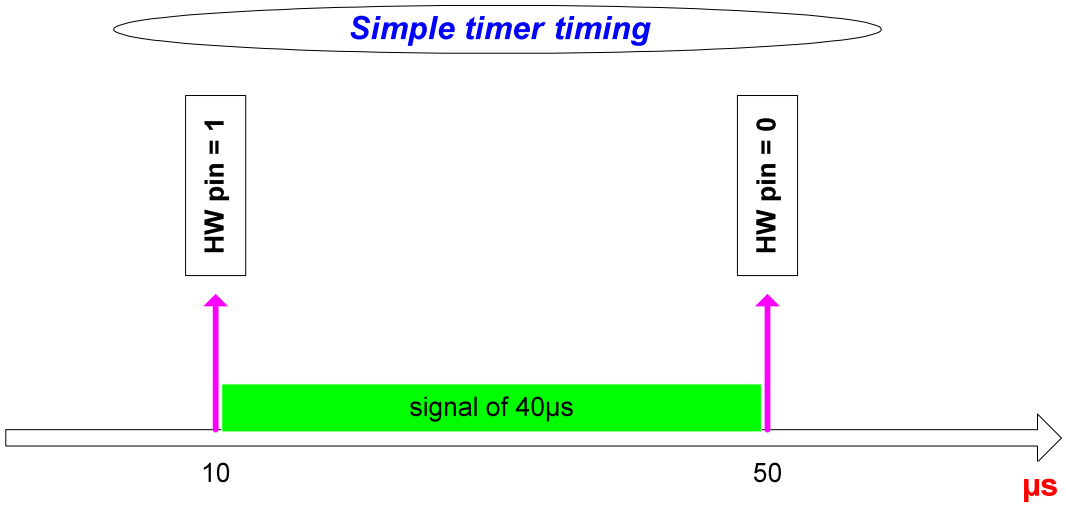
- Cyclic timer flags counting as multiples of 4ms:
    1- u8Flag200ms counts till 50 indicating that 200±10ms is finished, u8Flag100ms counts till 25 indicating that 100ms±10ms is finished.
    2- u16Flag10sec counts till 25,00 indicating that 10 seconds is finished (short circuit finished).
    3- u8Flag32ms counts till 8 indicating that 32ms is reached and it is the time to refresh the watchdog.
    4- u8TempCounter counts till 17 indicating that 72ms±4ms is reached, u8BatVolCounter counts till 5 indicating that 24ms±4ms is reached, and u16GatCounter counts till 30,000 indicating that 2 minutes is reached.

    Note that these flags will be updated be the scheduled tasks that enabled by the cyclic timer interrupt service routine using the flag u8MainFlag.

The figure below illustrates cyclic timer flags counting as multiples of 4ms:

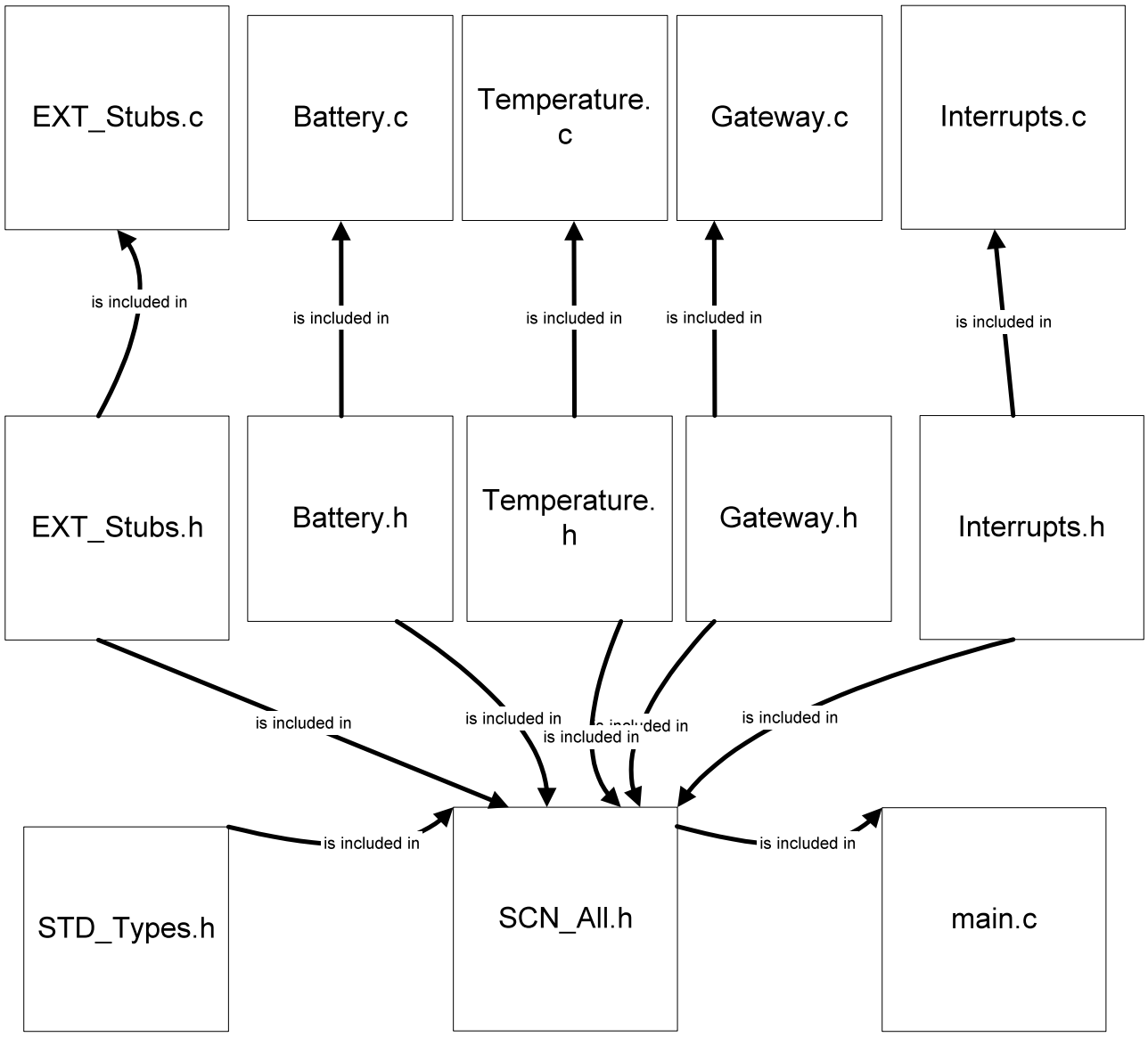


- For simple timer timing, it will take 40µs (80µs with 50% duty cycle) as a parameter and counts till it is finished (as illustrated in the figure below).



6.4 Functionality

File architecture:



Functionalities:

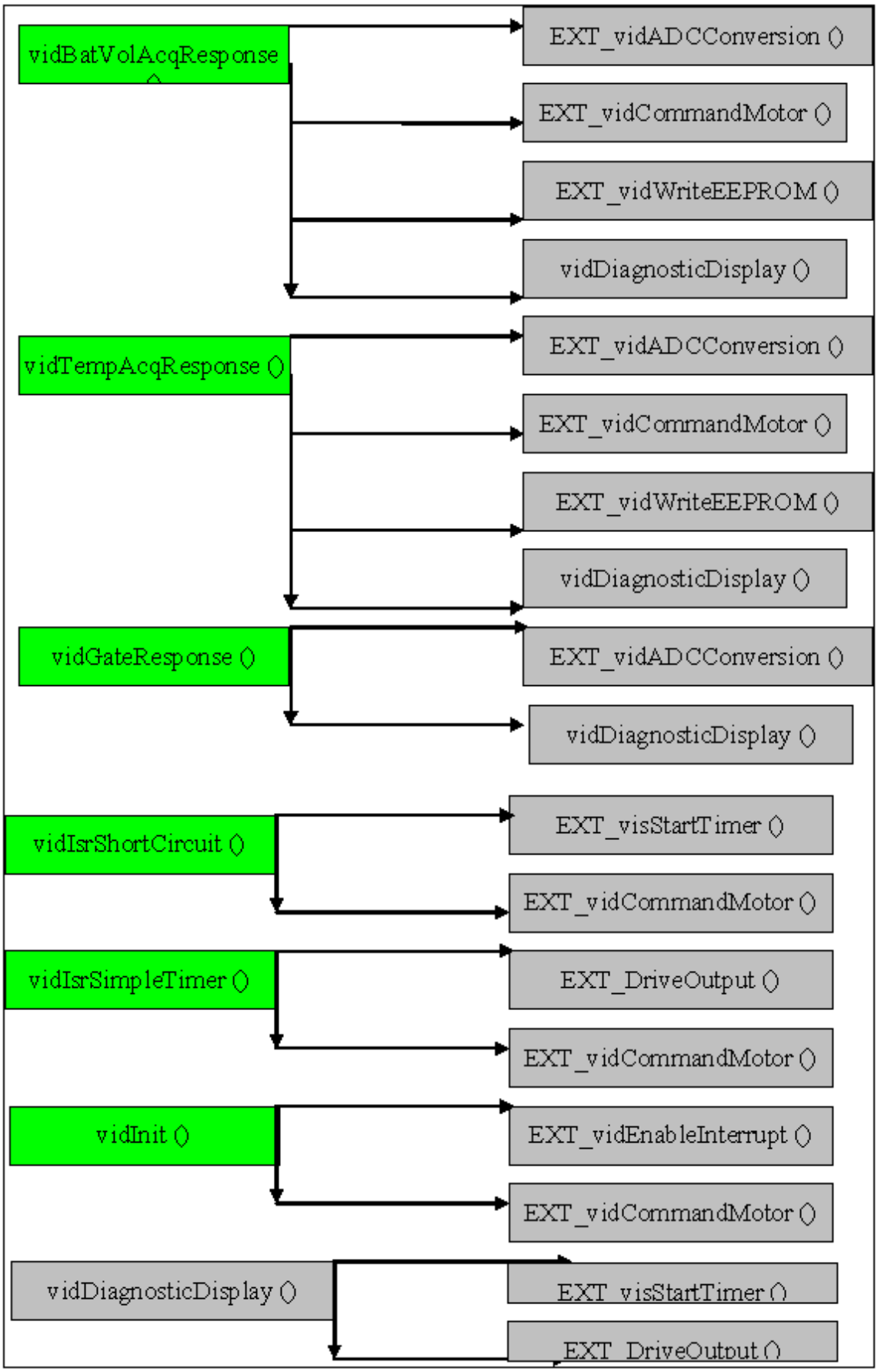
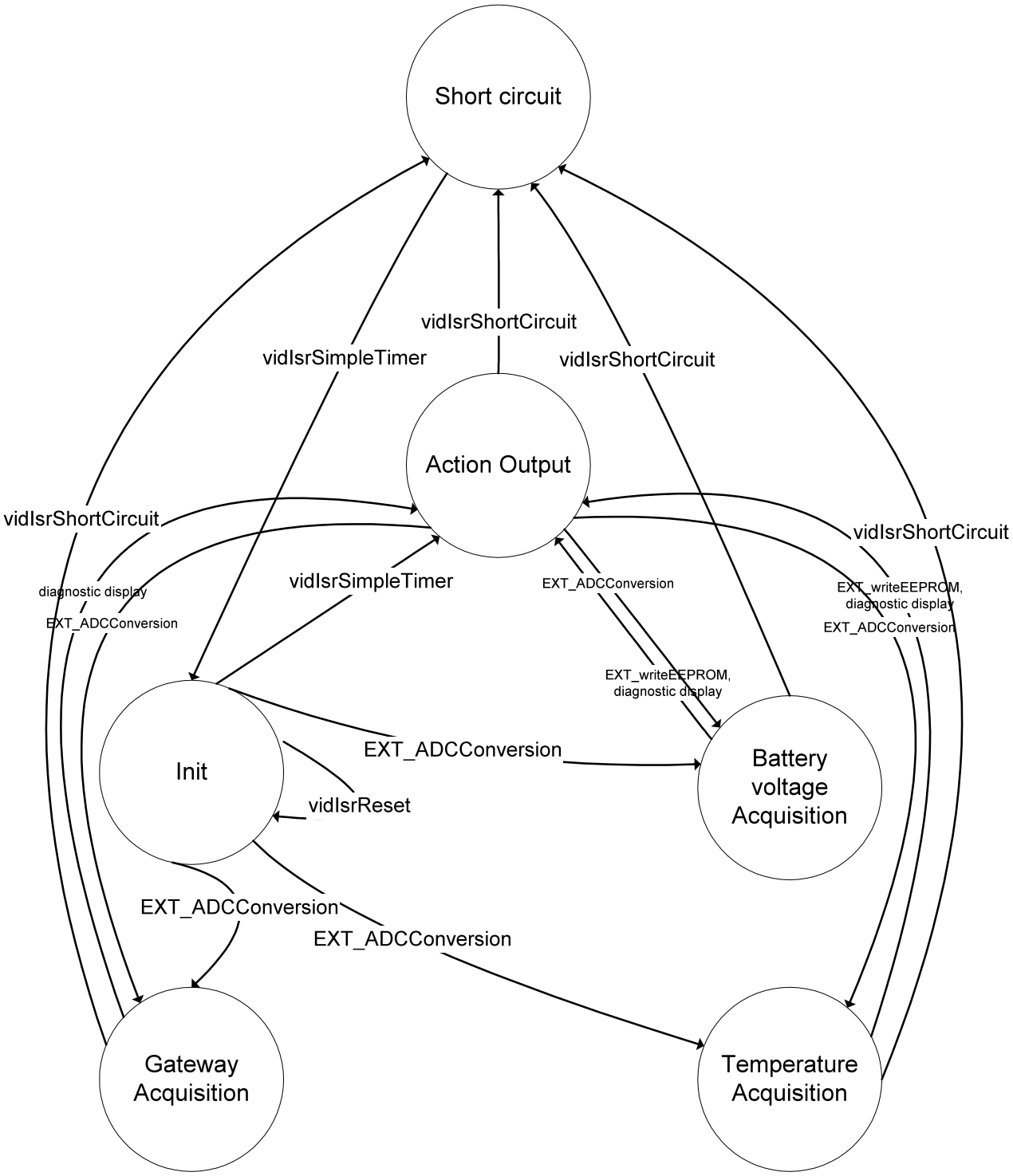


Diagram to illustrate the functionality:



7 Unit Testing

7.1 Functions and Modules test

Function SCN_vidBatVolAcqResponse

Test Specifications

Variable Name	Input/Output	Valid Equivalence Classes	Invalid Equivalence Classes	Values	Test Cases

Stubs analysis

Stub Name	Call Number	Parameter	Value	Test Cases
EXT_u16ADCConversion	1	I/P Par_1		
		O/P Par		
	2	I/P Par_1		
		O/P Par		
	3	I/P Par_1		
		O/P Par		
	4	I/P Par_1		
		O/P Par		
EXT_vidCommandMotor	1	I/P Par_1	1	2,3,9,11,12
EXT_vidWriteInEEPROM	1	I/P Par_1	0x00	2,3,9,11,12
		I/P Par_2	1	2,3,9,11,12
		I/P Par_3	0xAA	2,3,11,12
			0x55	9
EXT_vidDriveOutput	1	I/P Par_1	0	11,12,13,14
		I/P Par_2	1	11,13
			0	12,14

Function SCN_vidCyclicTimer

Test Specifications

Variable Name	Input/Output	Valid Equivalence Classes	Invalid Equivalence Classes	Values	Test Cases
SCN_u16Short CircuitFlag	input	[0]		0	1
		[1]		1	2,3
			[2..255]	2	N/A
				128	N/A
				255	N/A
LOC_u16Seco ndsFlag	input	[0..2500]		0	1
				500	2
				2499	3
				2500	N/A
			[2501..65535]	2501	N/A
				34018	N/A
				65535	N/A
	output	[0..2500]		0	1,3
				501	2
				2500	N/A
			[2501..65535]	2501	N/A
				34018	N/A
				65535	N/A
SCN_u8MainFl					

			[0]		
			[2..255]	2	N/A
				128	N/A
				255	N/A

Stubs analysis

Stub Name	Call Number	Parameter	Value	Test Cases
EXT_vidCommandMotor	1	I/P Par_1	0	3

Function SCN_vidGateAcqResponse testing procedure

Test Specifications

Variable Name	Input/Output	Valid Equivalence Classes	Invalid Equivalence Classes	Values	Test Cases

Stubs analysis

Stub Name	Call Number	Parameter	Value	Test Cases
EXT_u16ADCConversion	1	I/P Par_1	1	1,2,3,4
		O/P Par	563	1
			562	2
			451	3
			452	4
EXT_vidStartTimer	1	I/P Par_1	40	5
EXT_vidDriveOutput	1	I/P Par_1	1	5
		I/P Par_2	1	5

Module SCN_Temperature testing procedure

Function SCN_vidInit

Test Specifications

Variable Name	Input/Output	Valid Equivalence Classes	Invalid Equivalence Classes	Values	Test Cases
SCN_u8MainFlag	output	[1]		1	1
			[0]	0	N/A
			[2..255]	2	N/A
				128	N/A
				255	N/A
SCN_enuGatewayOnOff	output	[1]		1	1
			[0]	0	N/A
			[2..255]	2	N/A
				128	N/A
				255	N/A

Stubs analysis

Stub Name	Call Number	Parameter	Value	Test Cases
EXT_vidCommandMotor	1	I/P Par_1	0	1
EXT_vidEnableInterrupt	1	N/A	N/A	1

Function SCN_vidTempAcqResponse

Test Specifications

Variable Name	Input/Output	Valid Equivalence Classes	Invalid Equivalence Classes	Values	Test Cases

Stubs analysis

Stub Name	Call Number	Parameter	Value	Test Cases
EXT_u16ADCConversion	1	I/P Par_1	2	2,3,4,5,6,7,8,9,10,11,12,13,14
		O/P Par	955	2,3
			900	4
			100	5,7,9,10,13,14
			101	6
			956	8
			957	11
			911	12
	2	I/P Par_1	2	2,3,4,5,6,7,8,9,10,11,12,13,14
		O/P Par	955	2,3
			900	4
			100	5,7,9,10,13,14
			101	6
			956	8
			957	11
			911	12
	3	I/P Par_1	2	2,3,4,5,6,7,8,9,10,11,12,13,14
		O/P Par	955	2,3
			900	4
			100	5,7,9,10,13,14
			101	6
			956	8

ndMotor		Par_1		
EXT_vidWriteInEEPROM	1	I/P Par_1	0x002	2,3,9,11
			0x02	12
		I/P Par_2	1	2,3,9,11,12
		I/P Par_3	0xAA	2,3,11,12
			0x55	9
EXT_vidDriveOutput	1	I/P Par_1	2	11,12,13,14
		I/P Par_2	1	11,13
			0	12,14

Module SCN_Watchdog testing procedure

Function SCN_vidWatchdog

Test Specifications

Variable Name	Input/Output	Valid Equivalence Classes	Invalid Equivalence Classes	Values	Test Cases
LOC_u8FlagThirtyTwoms	input	[8]	N/A	8	N/A
		[21]	N/A	21	N/A
		[34]	N/A	34	N/A
		[47]	N/A	47	N/A
		[60]	N/A	60	N/A
		[73]	N/A	73	N/A
		[86]	N/A	86	N/A
		[99]	N/A	99	N/A
		[112]	N/A	112	N/A
		[125]	N/A	125	N/A

		[151]	N/A	151	N/A
		[164]	N/A	164	N/A
		[177]	N/A	177	N/A
		[190]	N/A	190	N/A
		[203]	N/A	203	N/A
		[216]	N/A	216	N/A
		[229]	N/A	229	N/A
		[242]	N/A	242	N/A
		[255]	N/A	255	N/A
	output	[0..255]	N/A	0	2,3,4,5
				1	1
				255	N/A

Stubs analysis

Stub Name	Call Number	Parameter	Value	Test Cases
EXT_vidWatchdogRefresh	1	N/A	N/A	2,3,4

7.2 Test Cases Description

Here should be the file scripts to run the unit and module testing execution for the test cases that defined in the previous tables, as an example:
SCN_vidBatVolAcqResponse.ptu
SCN_vidCyclicTimer.ptu
SCN_vidGateAcqResponse.ptu
SCN_vidInit.ptu
SCN_vidShortCircuit.ptu
SCN_vidTempAcqResponse.ptu
SCN_vidWatchdog.ptu

7.3 Structural Coverage Analysis

SCN_vidBatVolAcqResponse	
Hit	yes
Functions and exits	100.0% (2/2)
Statement blocks	100.0% (1/1)
Implicit blocks	none
Decisions	100.0% (1/1)
Loops	none
Basic conditions	none
Modified conditions	none
Multiple conditions	none

SCN_vidCyclicTimer	
Hit	yes
Functions and exits	100.0% (2/2)
Statement blocks	100.0% (1/1)
Implicit blocks	none
Decisions	100.0% (1/1)
Loops	none
Basic conditions	none
Modified conditions	none
Multiple conditions	none

SCN_vidGateAcqResponse	
Hit	yes
Functions and exits	100.0% (2/2)
Statement blocks	100.0% (1/1)
Implicit blocks	none
Decisions	100.0% (1/1)
Loops	none
Basic conditions	none
Modified conditions	none
Multiple conditions	none

SCN_vidShortCircuit	
Hit	yes
Functions and exits	100.0% (2/2)
Statement blocks	100.0% (1/1)
Implicit blocks	none
Decisions	100.0% (1/1)

Multiple conditions	none

SCN_vidTempAcqResponse	
Hit	yes
Functions and exits	100.0% (2/2)
Statement blocks	100.0% (1/1)
Implicit blocks	none
Decisions	100.0% (1/1)
Loops	none
Basic conditions	none
Modified conditions	none
Multiple conditions	none

SCN_vidWatchdog	
Hit	yes
Functions and exits	100.0% (2/2)
Statement blocks	100.0% (1/1)
Implicit blocks	none
Decisions	100.0% (1/1)
Loops	none
Basic conditions	none
Modified conditions	none
Multiple conditions	none

7.4 Unit Test Results

Module SCN_BatVol test results

Function	Test Result	Functional Coverage	Structural Coverage	Technical Fact Issued
SCN_vidBatVolAcqResponse	OK	100.0%	100.0%	None

Module SCN_Interrupts test results

Function	Test Result	Functional Coverage	Structural Coverage	Technical Fact Issued
SCN_vidCyclicTimer	OK	100.0%	100.0%	None
SCN_vidShortCircuit	OK	100.0%	100.0%	None

Function	Test Result	Functional Coverage	Structural Coverage	Technical Fact Issued
SCN_vidGateAcqResponse	OK	100.0%	100.0%	None

Module SCN_Temperature test results

Function	Test Result	Functional Coverage	Structural Coverage	Technical Fact Issued
SCN_vidInit	OK	100.0%	100.0%	None
SCN_vidTempAcqResponse	OK	100.0%	100.0%	None

Module SCN_Watchdog test results

Function	Test Result	Functional Coverage	Structural Coverage	Technical Fact Issued
SCN_vidWatchdog	OK	100.0%	100.0%	None

8 Conclusion

Through the detailed case study (Light control system), it seems that the integration between software and electronics hardware as part of system engineering to address CMMI is doable through various steps. The steps involves thinking about the software and electronics hardware at the same time during development (V-cycle) which normally creates new phases beside detailed design and unit testing like real time analysis and architectural design which is exactly the main point for addressing part of system engineering in CMMI (part of evolution engineering). The suggestion is to add Real Time Analysis, Architectural Design in software evolution and correction. As a result, Software evolution and correction will involve:

- Detailed design
- Real time analysis
- Architectural design
- Correction
- Testing
- documentation

9 References

- VIAS (Valeo Interbranch Automotive Software)
- Prof. Alain, April