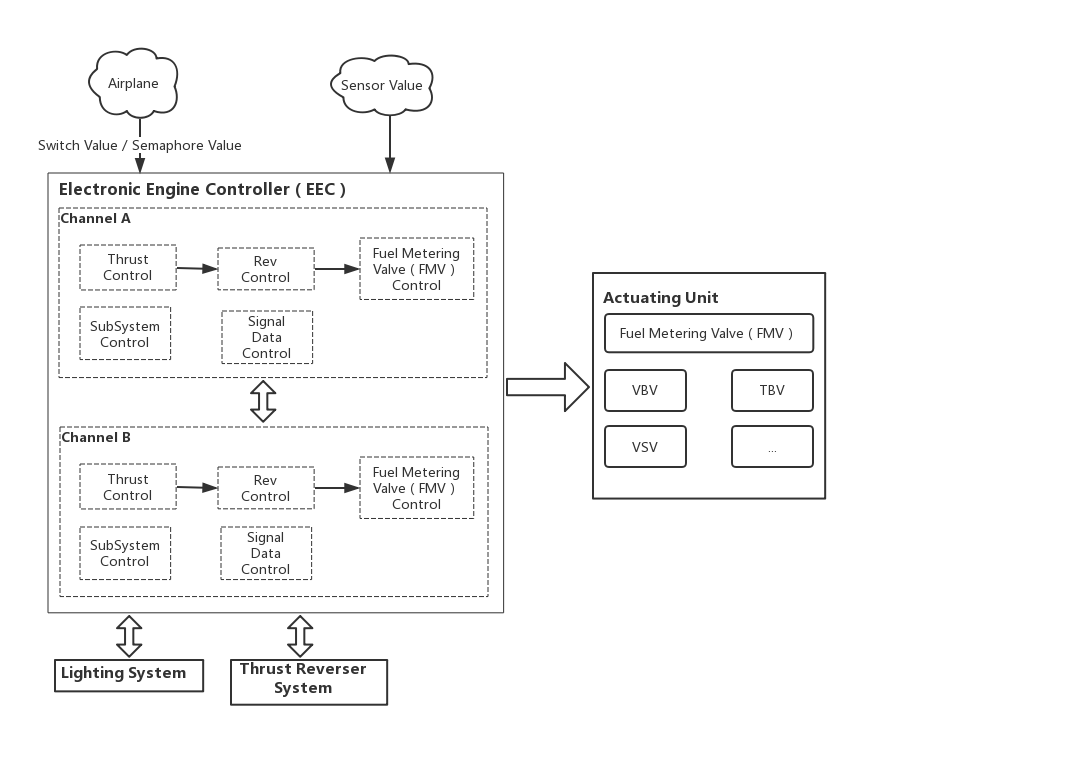
# Specification of an Airplane Engine Control Software

**1. Brief introduction to the airplane engine control software**

A full authority digital engine (or electronics) control (FADEC) is a system consisting of a digital computer, called an "electronic engine controller" (EEC), and its related accessories that control all aspects of aircraft engine performance.

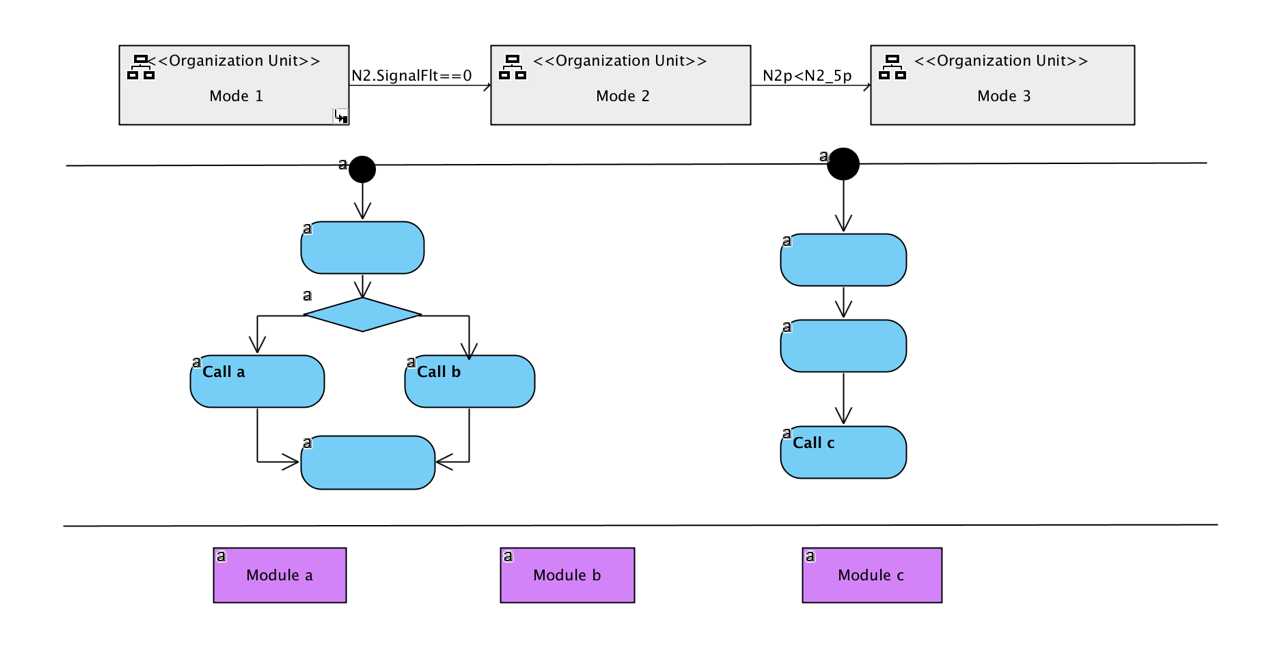
The overview of the FADEC system as shown in the Figure 1. FADEC works by receiving multiple input variables of the current flight condition including air density, throttle lever position, engine temperatures, engine pressures, and many other parameters. The inputs are received by the EEC and analyzed up to 100 times per second. Engine operating parameters such as fuel flow, stator vane position, air bleed valve position, and others are computed from this data and applied as appropriate. FADEC also controls engine starting and restarting. The FADEC's basic purpose is to provide optimum engine efficiency for a given flight condition.

****

**Figure 1: The overview of the FADEC system**

EEC which called airplane engine control software is the main component of FADEC. It controls the engine actuation system depending upon the situation and the commands received from cockpit. It receives data from all the sensors, commands (Switch value or Semaphore Value) from cockpit. It processes the data and figures out what control commands need to be given to the Actuating Unit and then execute them.

When the system stays in the certain mode, it needs periodic perform the specific function modules in a certain cycle. When the guard of the mode transition is satisfied, the mode of the system will transfer to the another mode. The overview of the control software as shown in Figure 2.



**Figure 2: The overview of the control software**

**2. Functional requirements of the airplane engine control software**

In this requirement, we have two types requirement, one is called Mode requirement, the other is called Module requirement. The system will only and if only stay in one mode of the Mode set.

The Mode set = {STARTABORT Mode, START Mode, SLOW Mode, BEYONDSLOW Mode, URGENSTOP Mode, STAND Mode, NORMAL Mode, WAIT Mode, OIL Mode, GROUNDSTART Mode}.

As the structure requirement shown below is the mode requirement of the START Mode. This requirement means that if the system stays in this mode, the system will periodically call the \_4.2.2.6.1.1 module in 5ms cycle. In the end time of cycle,

* if the guard of the Mode Transition “PLA.SelfFlt ==true” is satisfied, the mode of system will transfer to the WAIT Mode,
* if the guard of the Mode Transition “PLA.BITSetTime==false” is satisfied, the mode of system will transfer to the BEYONDSLOW Mode,
* if any guard of the Mode Transition is not satisfied, the mode of system will still stay in the START Mode in the next cycle.

**Mode Name：**START Mode

**PreCondition：**ES== ES\_START

**Initialization：**;

**ControlFlow：**

**PeriodTask**：5(ms)

{

Call(\_4.2.2.6.1.1);

}

**ModeTransition：**

**1Priority**: 1

**Description**：Enter Stand Mode

**Guard**：PLA.SelfFlt ==true

**Action**：{ ES= ES\_WAIT;}

**2Priority**: 1

**Description**：Enter Beyond Slow Mode

**Guard**：PLA.BITSetTime==false

**Action**：{ ES= ES\_BEYONDSLOW;}

As the structure requirement shown below is the mode requirement of the STAND Mode. This requirement means that if the system stays this mode, the system will periodically call the \_4.2.2.6.1.1 module in 5ms cycle. In the end time of cycle,

* if the guard of the Mode Transition “N2.SignalFlt==true” is satisfied, the mode of system will transfer to the WAIT Mode,
* if any guard of the Mode Transition is not satisfied, the mode of system will still stay in the STAND Mode in the next cycle.

**Mode Name：**STAND\_Mode

**PreCondition：**ES==ES\_ STAND

**Initialization：**;

**ControlFlow：**

**PeriodTask**：5(ms)

{

Call(\_4.2.2.6.1.1);

}

**ModeTransition：**

**1Priority**: 1

**Description**：Enter Stand Mode

**Guard**：N2.SignalFlt==true

**Action**：{ ES= ES\_WAIT;}

As the structure requirement shown below is the mode requirement of the SLOW Mode. This requirement means that if the system stays in this mode, the system will periodically call the \_4.2.2.6.1.1 module in 5ms cycle. In the end time of cycle,

* if the guard of the Mode Transition “N2p<500” is satisfied, the mode of system will transfer to the BEYONDSLOW Mode,
* if the guard of the Mode Transition “N2p>200” is satisfied, the mode of system will transfer to the NORMAL Mode,
* if any guard of the Mode Transition is not satisfied, the mode of system will still stay in the SLOW Mode in the next cycle.

**Mode Name：**SLOW\_Mode

**PreCondition：**ES==ES\_SLOW

**Initialization：**;

**ControlFlow：**

**PeriodTask**：5(ms)

{

Call(\_4.2.2.6.1.1);

}

**ModeTransition：**

**1Priority**: 1

**Description**：Enter Beyond Slow Mode

**Guard**：N2p<500

**Action**：{ ES= ES\_BEYONDSLOW;}

**2Priority**: 1

**Description**：Enter Normal Mode

**Guard**：N2p>200

**Action**：{ ES= ES\_NORMAL;}

As the structure requirement shown below is the mode requirement of the BEYONDSLOW Mode. This requirement means that if the system stays in this mode, the system will periodically call the \_4.2.2.6.1.1 module in 5ms cycle. In the end time of cycle,

* if the guard of the Mode Transition “ChanSts==true” is satisfied, the mode of system will transfer to the URGENSTOP Mode,
* if the guard of the Mode Transition “ChanSts==false” is satisfied, the mode of system will transfer to the SLOW Mode,
* if any guard of the Mode Transition is not satisfied, the mode of system will still stay in the BEYONDSLOW Mode in the next cycle.

**Mode Name：**BEYONDSLOW\_Mode

**PreCondition：**ES==ES\_BEYONDSLOW

**Initialization：**;

**ControlFlow：**

**PeriodTask**：5(ms)

{

Call(\_4.2.2.6.1.1);

}

**ModeTransition：**

**1Priority**: 1

**Description**：Enter urgen stop Mode

**Guard**：ChanSts==true

**Action**：{ ES= ES\_URGENSTOP;}

**2Priority**: 1

**Description**：Enter Slow Mode

**Guard**：ChanSts==false

**Action**：{ ES= ES\_SLOW;}

As the structure requirement shown below is the mode requirement of the WAIT Mode. This requirement means that if the system stays in this mode, the system will periodically call the \_4.2.2.6.1.1 module in 5ms cycle. In the end time of cycle,

* if the guard of the Mode Transition “WheelLoadSignal==1” is satisfied, the mode of system will transfer to the OIL Mode,
* if any guard of the Mode Transition is not satisfied, the mode of system will still stay in the WAIT Mode in the next cycle.

**Mode Name：**WAIT\_Mode

**PreCondition：**ES==ES\_WAIT

**Initialization：**;

**ControlFlow：**

**PeriodTask**：5(ms)

{

Call(\_4.2.2.6.1.1);

}

**ModeTransition：**

**1Priority**: 1

**Description**：Enter Oil Mode

**Guard**：WheelLoadSignal==1

**Action**：{ ES= ES\_OIL;}

As the structure requirement shown below is the mode requirement of the OIL Mode. This requirement means that if the system stays in this mode, the system will periodically call the \_4.2.2.6.1.1 module in 5ms cycle. In the end time of cycle,

* if the guard of the Mode Transition “N2\_5p>1000” is satisfied, the mode of system will transfer to the STARTABORT Mode,
* if any guard of the Mode Transition is not satisfied, the mode of system will still stay in the OIL Mode in the next cycle.

**Mode Name：**OIL\_Mode

**PreCondition：**ES==ES\_OIL

**Initialization：**;

**ControlFlow：**

**PeriodTask**：5(ms)

{

Call(\_4.2.2.6.1.1);

}

**ModeTransition：**

**1Priority**: 1

**Description**：Enter Startabort Mode

**Guard**：N2\_5p>1000

**Action**：{ ES= ES\_STARTABORT;}

As the structure requirement shown below is the mode requirement of the STARTABORT Mode. This requirement means that if the system stays in this mode, the system will periodically call the \_4.2.2.6.1.1 module in 5ms cycle. In the end time of cycle,

* if the guard of the Mode Transition “WfDem>30” is satisfied, the mode of system will transfer to the URGENSTOP Mode,
* if any guard of the Mode Transition is not satisfied, the mode of system will still stay in the STARTABORT Mode in the next cycle.

**Mode Name：**STARTABORT\_Mode

**PreCondition：**ES==ES\_STARTABORT

**Initialization：**;

**ControlFlow：**

**PeriodTask**：5(ms)

{

Call(\_4.2.2.6.1.1);

}

**ModeTransition：**

**1Priority**: 1

**Description**：Enter Startabort Mode

**Guard**：WfDem>30

**Action**：{ ES= ES\_URGENSTOP;}

As the structure requirement shown below is the mode requirement of the GROUNDSTART Mode. This requirement means that if the system stays in this mode, the system will periodically call the \_4.2.2.6.1.1 module in 5ms cycle. In the end time of cycle,

* if the guard of the Mode Transition “FUELCTRL == 0” is satisfied, the mode of system will transfer to the SLOW Mode,
* if the guard of the Mode Transition “FUELCTRL == 1” is satisfied, the mode of system will transfer to the STARTABORT Mode,
* if any guard of the Mode Transition is not satisfied, the mode of system will still stay in the GROUNDSTART Mode in the next cycle.

**Mode Name：**GROUNDSTART\_Mode

**PreCondition：**ES== ES\_GROUNDSTART

**Initialization：**;

**ControlFlow：**

**PeriodTask**：5(ms)

{

Call(\_4.2.2.6.1.1);

}

**ModeTransition：**

**1Priority**: 1

**Description**：Enter Slow Mode

**Guard**：FUELCTRL == 0

**Action**：{ ES= ES\_SLOW;}

**2Priority**: 1

**Description**：Enter Startabort Mode

**Guard**：FUELCTRL == 1

**Action**：{ ES= ES\_STARTABORT;}

As the structure requirement shown below is the mode requirement of the NORMAL Mode. This requirement means that if the system stays in this mode, the system will periodically call the \_4.2.2.6.1.1 and the \_4.2.3.4.1.10 module in 20ms cycle. In the end time of cycle,

* if the guard of the Mode Transition “P3b.SignalFlt==1” is satisfied, the mode of system will transfer to the WAIT Mode,
* if any guard of the Mode Transition is not satisfied, the mode of system will still stay in the NORMAL Mode in the next cycle.

**Mode Name：** NORMAL\_Mode

**PreCondition**： ES== ES\_NORMAL

**Initialization**： ;

**ControlFlow**：

**PeriodTask**：20(ms)

{

Call(\_4.2.2.6.1.1);

Call(\_4.2.3.4.1.10);

}

**ModeTransition**：

**1Priority**: 1

**Description：**Enter Stand Mode

**Guard**：P3b.SignalFlt==1

**Action**：{ES = ES\_WAIT;}

As the structure requirement shown below is the mode requirement of the URGENSTOP Mode. This requirement means that if the system stays in this mode, the system will periodically call the \_4.2.2.6.1.1 module in 5ms cycle. In the end time of cycle,

* if the guard of the Mode Transition “N2p<N2\_5p||(N2.SignalFlt ==1&&(duration(EmergencyStopTimerThsld,s, N2.SignalFlt==1)))” is satisfied, the mode of system will transfer to the WAIT Mode,
* if any guard of the Mode Transition is not satisfied, the mode of system will still stay in the URGENSTOP Mode in the next cycle.

**Mode Name：** URGENSTOP\_Mode

**PreCondition**： ES==ES\_URGENSTOP

**Initialization**： ;

**ControlFlow**：

**PeriodTask**：5(ms)

{

Call(\_4.2.2.6.1.1);

}

**ModeTransition**：

1**Priority**: 1

**Description**：Enter Stand Mode

**Guard**：N2p<N2\_5p||(N2.SignalFlt ==1&&(duration(EmergencyStopTimerThsld,s, N2.SignalFlt==1)))

**Action**：{ ES = ES\_WAIT;}

As the structure requirement shown below is the module requirement of the Reconstruct Signal N2

. This requirement means that if the system receive the N2 value from the sensor, it will reconstruct signal value before the system use it.

**Task Name**：Reconstruct Signal N2

**Task Number**：\_4.2.2.6.1.1

**Description**：None

**Task PreCondition**：1

**Task Input**：N2.SensorFlt, Nac.VoteData

**Task Output**：N2.ValidData

**Task ControlFlow**：

if(N2.SensorFlt == FAULT&& Nac.SensorFlt == NOFAULT&& Nac.VoteData<1)

{

if((N2\_1 -Nac\_1) < ModelLimit \* N2Design)

{

N2.ValidData = Nac.VoteData+ N2\_1;

N2.SignalFlt = NOFAULT;

if(N2.SensorFlt == NOFAULT && Nac.SensorFlt == NOFAULT)

{

N2.ValidData = N2\_1;

N2. SignalFlt = FAULT;

}

}

if((N2\_1 -Nac\_1\*NacVsN2)< ModelLimit)

{

N2.ValidData = Nac.VoteData+ N2\_1;

N2.SignalFlt = NOFAULT;

}

}

if(N2.SensorFlt == FAULT && Nac.SensorFlt == FAULT&&N2r25p>2)

{

N2.ValidData = N2\_1;

N2. SignalFlt = FAULT;

}

if(Nac.SensorFlt == NOFAULT && (N2\_1 - Nac\_1)>= ModelLimit \* N2Design)

{

N2.ValidData = N2\_1;

N2. SignalFlt = FAULT;

}

As the structure requirement shown below is the module requirement of the Reconstruct Signal P3b

. This requirement means that if the system receive the P3b value from the sensor, it will reconstruct signal value before the system use it.

**Task Name**：Reconstruct Signal P3b

**Task Number**：\_4.2.2.6.1.2

**Description**：None

**Task PreCondition**：1

**Task Input**：P3b.SensorFlt

**Task Output**：P3b.ValidData

**Task ControlFlow**：

N2.SensorFlt= atan(0.5);

N2.SensorFlt= last(ChanSts);

N2.SensorFlt= last(ChanSts,2);

if(first()){

N2.SensorFlt=power(9,0.5);

}

N2.SensorFlt=sat((5+7)/2)+abs(2);

N2.SensorFlt=sat((5-8)/2);

N2.SensorFlt=sat((5+7)/14);

N2.SensorFlt=sigma(1,totalperiod,last(Nac\_1)+Nac\_1);

N2.SignalFlt=sqrt(FAULT);

if( P3b.SensorFlt == FAULT && Ps3.SensorFlt == NOFAULT)

{

P3b.ValidData = Ps3.VoteData;

P3b.SignalFlt = NOFAULT;

}

if( P3b.SensorFlt == FAULT && Ps3.SensorFlt == FAULT)

{

P3b.SignalFlt = FAULT;

}

else

{

if( P3b.SensorFlt == A)

{

P3b.SignalFlt = FAULT;

}

else

{

P3b.SignalFlt = FAULT;

}

}

if( P3b.SensorFlt == FAULT && Ps3.SensorFlt == FAULT)

{

P3b.SignalFlt = FAULT;

}

As the structure requirement shown below is the module requirement of the Continuous Time Signal. This requirement means that if the system receive the value Continuous Time Signal, the system will handle the signal value According to the following process.

**Task Name**：Continuous Time Signal

**Task Number**：\_1.2.1.2.2

**Description**：None

**Task PreCondition**：1

**Task Input**：ChanSts, CCDLFlt, XX.PtnrFlt, XX.PtnrDiagData, XX.SelfFlt, XX.DiagData, XX.WarpLimit, XX.WarpLorH

**Task Output**：XX.VoteData, XX.SensorFlt, XX.OTFlt

**Task ControlFlow**：

if( ChanSts == 1 && XX.SelfFlt == 0)

{

XX.VoteData = XX.DiagData;

XX.SensorFlt = NOFAULT;

XX.OTFlt = NOFAULT;

}

if( ChanSts == 1 && XX.SelfFlt == 1 && CCDLFlt == 0 &&XX.PtnrFlt == 0)

{

XX.VoteData = XX.PtnrDiagData;

XX.SensorFlt = NOFAULT;

XX.OTFlt = NOFAULT;

}

if(XX.FirstFlag == 0)

{

XX.VoteData = XX.PreVoteData;

XX.SensorFlt = FAULT;

XX.OTFlt = NOFAULT;

}

if(XX.FirstFlag == 1)

{

XX.VoteData = XX.InitVal;

XX.SensorFlt = FAULT;

}

if( ChanSts == 0 && XX.SelfFlt == 1 && CCDLFlt == 0 &&XX.PtnrFlt == 0)

{

XX.VoteData = XX.PtnrDiagData;

XX.SensorFlt = NOFAULT;

}

if( ChanSts == 0 && (CCDLFlt == 1 || XX.PtnrFlt ==1)&&XX.SelfFlt == 0)

{

XX.VoteData = XX.DiagData;

XX.SensorFlt = NOFAULT;

}

if( ChanSts == 0 && (CCDLFlt == 1 || XX.PtnrFlt ==1)&&XX.SelfFlt == 1 && XX.FirstFlag == 0)

{

XX.VoteData = XX.PreDiagData;

XX.SensorFlt = FAULT;

}

if( ChanSts == 0 && (CCDLFlt == 1 || XX.PtnrFlt ==1)&&XX.SelfFlt == 1 && XX.FirstFlag == 1)

{

XX.VoteData = XX.InitVal;

XX.SensorFlt = FAULT;

}

if(abs(XX.DiagData-XX.PtnrDiagData)<= XX.WarpLimit)

{

XX.OTFlt = NOFAULT;

}

if(XX.SelfFlt == 0

&& XX.PtnrFlt ==0

&& CCDLFlt == 0

&& abs(XX.DiagData-XX.PtnrDiagData)> XX.WarpLimit)

{

XX.OTFlt = FAULT;

}

if(XX.SelfFlt==1 || XX.PtnrFlt==1 || CCDLFlt==1 )

{

XX.OTFlt = NOFAULT;

}

As the structure requirement shown below is the module requirement of the Signal T25 Process. This requirement means that if the system receive the T25 value from the sensor, the system will handle the signal value according to the following process.

**Task Name**：Signal T25 Process

**Task Number**：\_1.2.1.2.3

**Description**：None

**Task PreCondition**：ADRef.Flt ==0

**Task Input**：ID\_AI\_V\_T25, T25.PtnrDiagData, T25.PtnrFlt, CCDLFlt, T25.DemarCurve, T25.BITSetTime, T25.BITClrTime, T25.ExtreSetTime, T25.ExtrClrTime, T25.SlopeSetTime, T25.FirstBITTime, T25.FirstSlopeTime, T25.FirstExtreTime, T25.BITMin, T25.BITMax, T25.ExtreMin, T25.ExtreMax, T25.SlopeLimit, T25.WarpLimit, T25.WarpLimit, T25.InitVal, T25.InerFilterCo, T25.T1, T25.T2, T25.StepLimit

**Task Output**：T25.SqlData, T25.MeanData, T25.DiagData, T25.VoteData, T25.FiltData, T25.LeadData, T25.ValidData, T25.BITInv, T25.BITFlt, T25.ExtreInv, T25.ExtreFlt, T25.SlopeInv, T25.SlopeFlt, T25.SelfFlt, T25.SensorFlt, T25.SignalFlt, T25.OTFlt

**Task ControlFlow**：

if(T25.BITFlt == 1 || T25.ExtreFlt == 1 || T25.SlopeFlt == 1)

{

T25.SelfFlt = 1;

}

else

{

T25.SelfFlt = 0;

}

if(T25.SelfFlt == 1 || T25.BITInv == 1 || T25.ExtreInv ==1 || T25.SlopeInv == 1 )

{

if(T25.FirstFlag == 0)

{

T25.DiagData = T25.DiagData;

}

else

{

T25.DiagData= T25.InitVal;

}

}

else

{

T25.DiagData = T25.MeanData;

}

T25.SignalFlt = T25.SensorFlt;

T25.ValidData = T25.VoteData;

if(T25.LeadData - T25.LastLeadData > T25.StepLimit)

{

T25.LeadData = T25.LastLeadData + T25.StepLimit;

}

else

{

if(T25.LastLeadData - T25.LeadData > T25.StepLimit)

{

T25.LeadData = T25.LastLeadData - T25.StepLimit;

}

}

As the structure requirement shown below is the module requirement of the Danger Alert about SWI15. This requirement means that if receive the switch value about SWI15, the system will perform according to the following process.

**Task Name**：Danger Alert about SWI15

**Task Number**：\_4.2.3.4.1.10

**Description**：None

**Task PreCondition**：1

**Task Input**：ES, SWI15,N2r25

**Task Output**：SWI15.flt

**Task ControlFlow**：

if(!duration(100,s, SWI15==1)) {

SWI15.flt =1;

}

if(ES==ES\_NORMALSTOP&& duration(1.2,s,SWI15==0))

{SWI15.flt =1;}

if(SWI15.flt ==1)

{SWI15.canClear =false;}

As the structure requirement shown below is the module requirement of the Calculate Angle. This requirement means that the system will calculate the output value according to the input value.

**Task Name**：Calculate Angle

**Task Number**：\_1.1

**Description**：None

**Task PreCondition**：1

**Task Input**：Ax, Ay, Az

**Task Output**：Wx, Wy, Wz

**Task ControlFlow**：

if(Ax>0 && Ay>0 && Az>0)

{

Wx = atan(Ax/sqrt(Ay\*Ay+Az\*Az))\*180/pi;

Wy = atan(Ay/sqrt(Ax\*Ax+Az\*Az))\*180/pi;

Wz = atan(Az/sqrt(Ax\*Ax+Ay\*Ay))\*180/pi;

}

As the structure requirement shown below is the module requirement of the Get Angular Velocity. This requirement means that the system will get the value from the sensor.

**Task Name**：Get Angular Velocity

**Task Number**：\_1.2

**Description**：get angular velocity

**Task PreCondition**：1

**Task Input**：Ax, Ay, Az

**Task Output**：Ax, Ay, Az

**Task ControlFlow**：

Ax = get(Ax);

Ay = get(Ay);

Az = get(Az);

As the structure requirement shown below is the module requirement of the Reconstruct Angular Velocity. This requirement means that the system will get the reconstruct the value before using it.

**Task Name**：Reconstruct Angular Velocity

**Task Number**：\_1.3

**Description**：None

**Task PreCondition**：1

**Task Input**：Ax, Ay, Az

**Task Output**：Ax, Ay, Az

**Task ControlFlow**：

Ax = Ax\*1.0;

Ay = Ay\*1.0;

Az = Az\*1.0;