DALLA MACCHINA A STATI ALLA MACCHINA

Autore: Enricomaria Pavan (e.pavan@beckhoff.it)

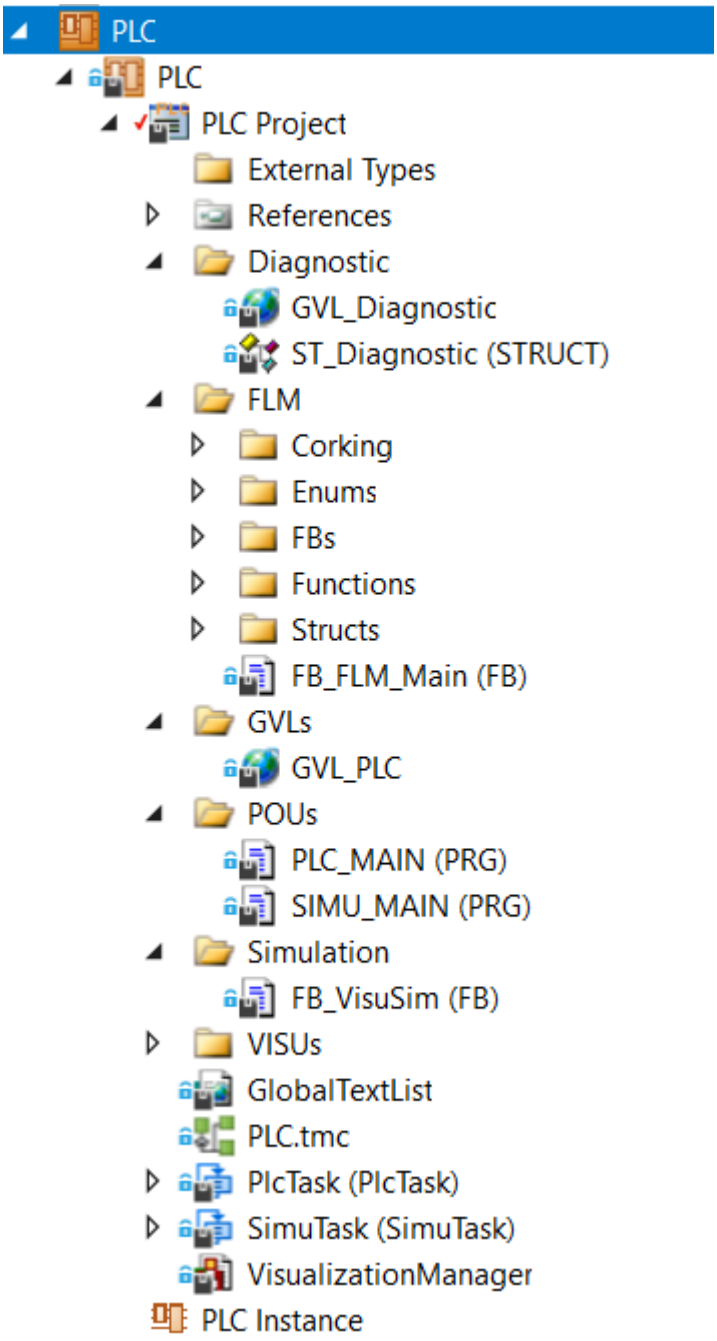
Data: 2023-04-05

1. Considerazioni iniziali

Il codice va organizzato e scomposto. I motivi sono molteplici:

- favorire la fase di test e di debug: più piccole sono le porzioni di codice da validare più la validazione è semplice, veloce e eventualmente automatizzabile. Più il programma è organizzato più agevole sarà isolare i difetti e risolverli
- usare le metodologie di programmazione più adatte allo scopo da raggiungere. Le macchine a stati sono utili e applicabili in moltissime situazioni ma non vanno bene ovunque e non sono utili ovunque
- sfruttare le caratteristiche dell'IE61131-3: in particolare la possibilità di usare linguaggi diversi in diversi. ST è il linguaggio che la fa sempre da padrone tuttavia per scopi particolari altri linguaggi possono essere molto utili: il LADDER in particolare e l'FBD

2. Organizzazione del programma



La cartella "Diagnostic" contiene la definizione del tipo di dato ST_Diagnostic e una lista di variabili globali al cui interno viene dichiarata solamente una costante. Questo tipo di dato è generico e non riferito alla macchina pertanto conviene metterlo in una cartella a parte. Se per esempio si decidesse di gestire due macchine dallo stesso programma questo tipo di dato verrebbe usato in entrambe

La cartella "FLM" contiene il codice della macchina. All'interno ci sono delle cartelle per contenere:

- FB
- Function
- Strutture
- Enumerazioni
- eventuali sotto funzionalità ("Corking"). Vale la pena di creare una cartella apposita se queste funzionalità vengono gestite con una macchina a stati (secondaria) e/o se prevedono tipi di dato (strutture e/o enumerazioni) dedicate

Si consiglia di usare una sigla di 3 o 4 lettere per identificare il tipo di macchina ("FLM" per FillerMachine) e si consiglia di usare la sigla per identificare i nomi delle funzioni, FB, strutture, enumerazioni. Una sigla è facile da memorizzare e permette di non appesantire il codice con ripetizioni continue di identiche parole molto lunghe.

Nella cartella POUs ci sono i PRG sono associati ai task uno ad uno: un PRG <-> un Task. Si consiglia di identificare con il nome la funzione del PRG:

- PLC_MAIN conterrà il programma delle macchine
- SIMU_MAIN conterrà il programma di simulazione (che ovviamente nella versione definitiva caricata in macchina non ci sarà)
- COMM_MAIN potrebbe contenere del codice necessario a gestire le comunicazioni verso l'esterno
- VISION_MAIN potrebbe contenere del codice per la gestione dei programmi di visione
- etc

Nella cartella GVLs ci sono i dati globali. Il consiglio è quello di avere una lista di dati globali per ogni Program (e quindi per ogni task ... 1 task <-> un PRG <-> una GVL). Se una GVL viene usata in più di un task, con le dovute attenzioni che non sono argomento di questo documento, il consiglio è di dichiararla nella GVL associata al task dove viene scritta. Se la GVL viene scritta da più task è il caso di chiedersi se le attenzioni dovute sono state prese perché probabilmente non è così.

3. La suddivisione del programma di macchina

3.1 Le strutture dati

Il programma della macchina si appoggia ad alcune strutture dati:

- ST_Diagnostic
- ST_FLM_Cmd
- ST_FLM_Ctrl
- ST_FLM_In
- ST_FLM_Motion
- ST_FLM_Out
- ST_FLM_Params
- ST_FLM_RawIn
- ST_FLM_RawOut
- ST_FLM_Status
- ST_FLM_StatusSaved

In ST_Diagnostic troviamo 5 array di booleani che identificano la presenza di eventi notevoli nella macchina:

- eventi critici
- errori
- segnalazioni
- informazioni
- messaggi

Le prime due categorie (eventi critici ed errori) identificano tipi di eventi per i quali è necessario arrestare la macchina.

```
TYPE ST_Diagnostic :
STRUCT
    aCritical : ARRAY[0..GVL_Diagnostic.cMAXEVENTS - 1] OF BOOL;    // eventi
critici
    aError : ARRAY[0..GVL_Diagnostic.cMAXEVENTS - 1] OF BOOL;       // errori
    aWarning : ARRAY[0..GVL_Diagnostic.cMAXEVENTS - 1] OF BOOL;     //
segnalazioni
    aInfo : ARRAY[0..GVL_Diagnostic.cMAXEVENTS - 1] OF BOOL;        //
informazioni
    aVerbose : ARRAY[0..GVL_Diagnostic.cMAXEVENTS - 1] OF BOOL;     // messaggi
END_STRUCT
END_TYPE
```

In ST_FLM_Cmd sono contenuti dei comandi utili per la macchina: start, stop, riconoscimento allarmi, comandi di jog degli assi, etc. Per lo più sono booleani che sono pensati per rimanere attivi per un singolo ciclo di PLC o, al massimo, per un breve tempo.

```
TYPE ST_FLM_Cmd :
STRUCT
    bAck                : BOOL; (* Riconoscimento allarmi *)
    bStop               : BOOL; (* Arresta la macchina *)
    bStart              : BOOL; (* Mette in marcia la macchina *)
END_STRUCT
END_TYPE
```

In ST_FLM_Ctrl sono contenuti dei booleani di controllo dell'esecuzioni della macchina: enable, emergenza, etc. Potrebbe sembrare uno spreco la definizione di una struttura per contenere solo un paio di booleani, ma vista l'importanza di questi è utile raggrupparli e distinguerli.

```
TYPE ST_FLM_Ctrl :
STRUCT
    bEmergency: BOOL;   // Emergenza. Deve essere TRUE perché la macchina possa
operare
    bEnable: BOOL;      // Abilitazione macchina. Deve essere TRUE perché la
macchina possa operare
END_STRUCT
END_TYPE
```

In ST_FLM_In e ST_FLM_OUT troviamo rispettivamente gli ingressi e le uscite fisiche della macchina. Gli elementi di queste strutture sono definiti con tipi di dati adatti all'elaborazione del programma (analogiche in REAL o LREAL e digitali eventualmente invertiti in modo da essere più comodi da usare) e riferiti ad unità ingegneristiche (nei programmi si cerchi sempre di usare unità di misura del S.I. https://it.wikipedia.org/wiki/Sistema_internazionale_di_unit%C3%A0_di_misura). Gli elementi di queste strutture **non** saranno linkati agli I/O.

```
TYPE ST_FLM_In :
STRUCT
    (* digital *)
    bEmergency : BOOL;                      // emergenza (NC)
    bStartButton : BOOL;                    // pulsante di start (NO)
    bStopButton : BOOL;                     // pulsante di stop (NO)
    bAckButton : BOOL;                      // pulsante di riconoscimento allarmi
(NO)
    bBottle : BOOL;                         // fotocellula presenza bottiglia (NO)
    bCorkingRotationLimitSwitch : BOOL;     // finecorsa rotazione tappatura (NO)
    (* analog *)
    fWaterLevel : REAL;     // livello acqua
    fMintLevel : REAL;      // livello menta
END_STRUCT
END_TYPE

TYPE ST_FLM_Out :
STRUCT
    bFixedYellow : BOOL;        // luce gialla fissa
    bBlinkingYellow: BOOL;      // luce gialla lampeggiante
    bFixedGreen : BOOL;         // luce verde fissa
    bBlinkingGreen : BOOL;      // luce verde lampeggiante
    bHorn : BOOL;               // sirena di segnalazione

    bConveyor : BOOL;           // nastro trasportatore

    bWater : BOOL;              // riempimento acqua
    bMint : BOOL;               // riempimento menta

    bCorkingVertical : BOOL;    // inserimento del tappo - movimento verticale
    bCorkingRotation : BOOL;    // inserimento del tappo - rotazione
END_STRUCT
END_TYPE
```

In ST_FLM_RawIn e ST_FLM_RawOut troviamo gli ingressi e le uscite fisiche della macchina come in ST_FLM_In e ST_FLM_OUT ma gli elementi in queste strutture sono definiti con tipi di dati adatti al link verso gli I/O ai quali saranno successivamente collegati. Da notare come la definizione di normalmente aperto e normalmente chiuso possano essere invertite rispetto alla struttura ST_FLM_In e ST_FLM_OUT.

```
TYPE ST_FLM_RawIn :
STRUCT
    (* digital *)
    bEmergency : BOOL;                      // emergenza (NC)
    bStartButton : BOOL;                    // pulsante di start (NO)
    bStopButton : BOOL;                     // pulsante di stop (NC)
    bAckButton : BOOL;                      // pulsante di riconoscimento allarmi
(NO)
    bBottle : BOOL;                         // fotocellula presenza bottiglia (NO)
```

```
    bCorkingRotationLimitSwitch : BOOL;     // finecorsa rotazione tappatura (NO)
    (* analog *)
    nWaterLevel : INT;                      // livello acqua
    nMintLevel : INT;                       // livello menta
END_STRUCT
END_TYPE

TYPE ST_FLM_RawOut :
STRUCT
    bConveyor : BOOL;           // nastro trasportatore
    bWater : BOOL;              // riempimento acqua
    bMint : BOOL;               // riempimento menta
    bYellow : BOOL;             // luce gialla
    bGreen : BOOL;              // luce verde
    bHorn : BOOL;               // sirena di segnalazione
    bCorkingVertical : BOOL;    // inserimento del tappo - movimento verticale
    bCorkingRotation : BOOL;    // inserimento del tappo - rotazione
END_STRUCT
END_TYPE
```

In ST_FLM_Params troviamo i parametri che definiscono il comportamento della macchina. L'istanza di questa struttura dovrà essere definita PERSISTENT o RETAIN in modo che i parametri vengano salvati e mantenuti alla riaccensione. Eventuali dispositivi secondari della macchina (il dispositivo di tappatura in questo caso) potrebbero avere le proprie strutture di parametri che devono essere inserite qui come istanza. È buona norma dare un valore di default agli elementi di questa struttura. I valori di default delle istanze di strutture di parametri dei dispositivi secondari dovranno però essere inseriti nella dichiarazione delle strutture secondarie.

```
TYPE ST_FLM_Params :
STRUCT
    nFillingCycles : UINT := 3;             // cicli di riempimento
    tWaterFillingTime : TIME := T#1S;       // durata riempimento acqua per ciclo
    tMintFillingTime : TIME := T#500MS;     // durata riempimento menta per ciclo
    fWaterMaxLevel : REAL := 1000.0;        // livello massimo letto dal sensore
(acqua)
    fMintMaxLevel : REAL := 200.0;          // livello massimo letto dal sensore
(menta)
    stCorking : ST_FLM_CorkingParams;       // parametri dispositivo di tappatura
END_STRUCT
END_TYPE
```

In ST_FLM_Status troviamo dati che descrivono lo stato della macchina. Sono utili sia per comunicazione interna (tra le varie parti che compongono il programma) sia per comunicazione esterna verso un supervisore o un HMI. Eventuali dispositivi secondari della macchina (il dispositivo di tappatura in questo caso) potrebbero avere le proprie strutture di stato che devono essere inserite qui come istanza.

```
TYPE ST_FLM_Status :
STRUCT

    stSaved : ST_FLM_StatusSaved;       // Valori ritenuti - utili p.e. per
statistiche
    bCritical : BOOL;                   // la macchina ha almeno una condizione
critica attiva
    bError : BOOL;                      // la macchina ha almeno una condizione di
errore attiva
    bWarning : BOOL;                    // la macchina ha almeno una segnalazione
attiva
    bInfo : BOOL;                       // la macchina ha almeno una informazione
attiva
    bVerbose : BOOL;                    // la macchina ha almeno un messaggio
attiva

    bFailure : BOOL;                    // la macchina è in avaria (critica +
error)
    eStep : E_FLM_Step;                 // passo della macchina a stati
    strStep : STRING;                   // passo come stringa
    tStepTime : TIME;                   // tempo di permanenza nel passo
    eErrorStep : E_FLM_Step;            // passo in cui si è generato l'errore
    nErrorID : DINT;                    // codice di errore

    stCorking : ST_FLM_CorkingStatus;   // struttura di stato del dispositivo di
tappatura

END_STRUCT
END_TYPE
```

In ST_FLM_StatusSaved troviamo delle variabili di stato della macchina che hanno la necessità di essere ritenuti. Si tratta, di solito, di valori statistici quali dei contatori dei cicli di avvio della macchina e contatori degli allarmi. In ogni caso qualsiasi valore di stato che abbia la necessità di essere ritenuto può essere aggiunto qui.

```
TYPE ST_FLM_StatusSaved :
STRUCT
nPowerOnCnt : UDINT;                // contatore accensioni macchina
nStartStopCnt : UDINT;              // contatore avvii macchina
nCCyclesCnt : UDINT;                // contatore cicli di riempimento
nCriticalCnt : UDINT;               // contatore eventi critici
nErrorCnt : UDINT;                  // contatore errori
nWarningCnt : UDINT;                // contatore segnalazioni
END_STRUCT
END_TYPE
```

3.2 I dati globali

```
VAR_GLOBAL PERSISTENT
    stFlmParams : ST_FLM_Params;
    stFlmStatusSaved : ST_FLM_StatusSaved;
END_VAR

VAR_GLOBAL
    stFlmDiagnostic : ST_Diagnostic;

    stFlmIn : ST_FLM_In;
    stFlmRawIn : ST_FLM_RawIn;
    stFlmOut : ST_FLM_Out;
    stFlmRawOut : ST_FLM_RawOut;
    stFlmCtrl : ST_FLM_Ctrl;
    stFlmStatus : ST_FLM_Status;
    stFlmMotion : ST_FLM_Motion;

END_VAR
```

I dati globali per lo più sono istanze delle strutture viste al paragrafo precedente. Le istanze saranno passate come ingressi o in_out all'FB principale della macchina oppure valorizzate dalle uscite della stessa. È da notare come le istanze della struttura Params e quella StatusSaved siano dichiarate "PERSISTENT". Tecnicamente alcune di queste variabili potevano essere dichiarate locali in PLC_MAIN tuttavia si preferisce dichiararle tutte qui per omogeneità in quanto alcune di esse sono ritentive e altre potrebbero venir scambiate con altre task.

3.3 Il Main (PLC_MAIN)

In PLC_MAIN troviamo una una parte tipica di inizializzazione e la chiamata all'FB principale della macchina. Sono inoltre presenti alcune chiamate per identificare l'indice del task su cui si sta girando e per recuperare il tempo di ciclo impostato e il tempo di esecuzione effettivo della task.

```
PROGRAM PLC_MAIN
VAR
    bInitialized : BOOL;

    fbGetCurTaskIndex : GETCURTASKINDEX;
    tTaskCycleTime : UDINT;                 // tempo di ciclo della task in
microsecondi
    tTaskExecTime : UDINT;                  // tempo di esecuzione della task in
microsecondi
    tTaskMaxExecTime : UDINT;               // tempo massimo di esecuzione della
task in microsecondi

    fbFlm : FB_FLM_Main;
END_VAR
```

```
IF NOT bInitialized THEN
    (* Get the Cycle Time of the task to which this PRG is assigned  *)
    fbGetCurTaskIndex();
    tTaskCycleTime := _TaskInfo[fbGetCurTaskIndex.index].CycleTime / 10;

ELSE

    (* task time *)
    tTaskExecTime := _TaskInfo[fbGetCurTaskIndex.index].LastExecTime / 10;
    IF tTaskExecTime > tTaskMaxExecTime THEN
            tTaskMaxExecTime := tTaskExecTime;
    END_IF;

    (* machine *)
    fbFlm(
        bEnable:= TRUE,
        tTaskCycleTime:= tTaskCycleTime,
        nId:= 0,
        bHwError:= FALSE,
        stRawIn := GVL_PLC.stFlmRawIn,
        stIn := GVL_PLC.stFlmIn,
        stRawOut := GVL_PLC.stFlmRawOut,
        stOut := GVL_PLC.stFlmOut,
        stCtrl := GVL_PLC.stFlmCtrl,
        stParams := GVL_PLC.stFlmParams,
        stDiagnostic := GVL_PLC.stFlmDiagnostic,
        stStatus := GVL_PLC.stFlmStatus,
        stStatusSaved := GVL_PLC.stFlmStatusSaved,
        stMotion := GVL_PLC.stFlmMotion,
    );

END_IF
```

3.4 Il Main di Macchina (FB_FLM_MAIN)

L'FB che gestisce la nostra macchina è un "main" di macchina che viene richiamato dal PLC_MAIN e si occupa di richiamare le FB specifiche, tra le quali la macchina a stati.

```
FUNCTION_BLOCK FB_FLM_Main

VAR_INPUT
    bEnable         : BOOL;             // abilitazione della FB
    tTaskCycleTime  : UDINT;            // tempo di ciclo della task
    nId             : USINT;            // id macchina
    bHwError        : BOOL;             // errore hardware
END_VAR

VAR_OUTPUT
```

```
END_VAR

VAR_IN_OUT
    stRawIn : ST_FLM_RawIn;                 // ingressi "crudi"
    stIn : ST_FLM_In;                       // ingressi
    stRawOut : ST_FLM_RawOut;               // uscite "crude"
    stOut : ST_FLM_Out;                     // uscite
    stCtrl : ST_FLM_Ctrl;                   // segnali di controllo
    stParams : ST_FLM_Params;               // parametri
    stDiagnostic : ST_Diagnostic;           // diagnostica
    stStatus : ST_FLM_Status;               // stati
    stStatusSaved : ST_FLM_StatusSaved;     // stati ritenuti
    stMotion : ST_FLM_Motion;               // struttura per motion

END_VAR

VAR
    bInitialized : BOOL;                    // inizializzazione
    stCmd : ST_FLM_Cmd;                     // comandi della macchina
    errno : INT;                            // codice di errore
    fbReadCmd : FB_FLM_ReadCmd;             // lettura dei comandi
    fbResetCmd : FB_FLM_ResetCmd;           // reset dei comandi
    fbReadCtrl : FB_FLM_ReadCtrl;           // lettura segnali di controllo
    fbReadIn : FB_FLM_ReadInputs;           // lettura ingressi
    fbDiagnostic : FB_FLM_Diagnostic;       // diagnostica
    fbStateMachine : FB_FLM_StateMachine;   // macchina a stati principale
    fbCorkingStateMachine : FB_FLM_CorkingStateMachine;
                                            // macchina a stati secondaria
    fbWriteOut : FB_FLM_WriteOutputs;       // scrittura uscita
END_VAR


IF NOT bInitialized THEN
    stStatus.stSaved := stStatusSaved;
    bInitialized := TRUE;
ELSE

    (* 01 - Leggi ingressi, controlli, parametri e comandi  *)
    fbReadIn( // ... );

    stIn := fbReadIn.stIn;

    fbReadCtrl( // ...);

    stCtrl := fbReadCtrl.stCtrl;

    fbReadCmd( // ...);

    stCmd := fbReadCmd.stCmd;
```

```
    (* 03 - Gestione eventi *)
    fbDiagnostic( // ...);

    stStatus.bCritical := fbDiagnostic.bCritical;
    stStatus.bError := fbDiagnostic.bError;
    stStatus.bWarning := fbDiagnostic.bWarning;
    stStatus.bInfo := fbDiagnostic.bInfo;
    stStatus.bVerbose := fbDiagnostic.bVerbose;
    stStatus.bFailure := fbDiagnostic.bFailure;

    stOut.bBlinkingYellow := fbDiagnostic.bBlinking;
    stOut.bFixedYellow := fbDiagnostic.bFixed;
    stOut.bHorn := fbDiagnostic.bHorn;

    (* 04 - Macchina a stati principale: *)
    fbStateMachine( // ...);

    stStatus.eStep := fbStateMachine.eStep;
    stStatus.strStep := TO_STRING(stStatus.eStep);
    stStatus.tStepTime := fbStateMachine.tStepTime;
    stStatus.eErrorStep := fbStateMachine.eErRstep;
    stStatus.nErrorID := fbStateMachine.nErrId;

    stOut.bConveyor := fbStateMachine.bConveyor;
    stOut.bWater := fbStateMachine.bWater;
    stOut.bMint := fbStateMachine.bMint;
    stOut.bBlinkingGreen := fbStateMachine.bBlinkingGreen;
    stOut.bFixedGreen := fbStateMachine.bFixedGreen;

    (* 05 - Macchine a stati secondarie *)
    fbCorkingStateMachine( // ...);

    stStatus.stCorking.eStep := fbCorkingStateMachine.eStep;
    stStatus.stCorking.tStepTime := fbCorkingStateMachine.tStepTime;
    stStatus.stCorking.eErrorStep := fbCorkingStateMachine.eErrStep;
    stStatus.stCorking.nErrorID := fbCorkingStateMachine.nErrID;

    stOut.bCorkingRotation := fbCorkingStateMachine.bRotation;
    stOut.bCorkingVertical := fbCorkingStateMachine.bVertical;

    (* 06 - Scrivi uscite e informazioni *)
    fbWriteOut( // ...);

    stStatusSaved := stStatus.stSaved;  // aggiorna struttura dati salvati

    (* 07 - fine - altre operazioni*)
    fbResetCmd();

END_IF;
```

3.5 Lettura ingressi e scrittura uscite

FB_FLM_ReadInputs legge gli ingressi scritti attraverso i link in ST_FLM_RawIn, li converte opportunamente e li scrive in ST_FLM_In.

FB_FLM_WriteOutputs legge le uscite scritte dal resto del programma in ST_FLM_Out, li converte opportunamente e li scrive in ST_FLM_RawOut da dove, per mezzo di link, saranno trasferiti alla periferia.

3.6 Lettura segnali di controllo e comandi

FB_FLM_ReadCtrl genera dei segnali di controllo che vengono usati dalle altre FB che compongono il programma della macchina. Si tratta principalmente di segnali di abilitazione generale e di gestione dell'emergenza (sicurezza)

FB_FLM_ReadCmd genera dei segnali di comando che vengono usati dalle altre FB che compongono il programma della macchina. In questo stato si tratta di segnali di start, stop, reset generali o particolari.

3.7 Diagnostica

FB_FLM_Diagnostic genera i segnali di Diagnostica che poi vengono usati nel resto del programma della macchina. In uscita dall'FB troviamo una struttura di diagnostica che mantiene la comunicazione di evento attivo anche nel caso in cui la causa scatenante non sia più attiva. In pratica affinché una comunicazione di evento venga azzerata è necessario che la causa non sia più attiva e che venga alzato il bit di riconoscimento degli eventi.

```
FUNCTION_BLOCK FB_FLM_Diagnostic
VAR_INPUT
    stIn : ST_FLM_In;
    stOut : ST_FLM_Out;
    stParams : ST_FLM_Params;
    stCtrl : ST_FLM_Ctrl;
    bHwError : BOOL;
    bAck : BOOL;                // riconoscimento degli eventi
    stStatus : ST_FLM_Status;
END_VAR

VAR_IN_OUT
    stMotion : ST_FLM_Motion;
    stDiagnostic : ST_Diagnostic;
END_VAR

VAR_OUTPUT
    bCritical : BOOL;
    bError : BOOL;
    bWarning : BOOL;
    bInfo  : BOOL;
    bVerbose : BOOL;
    bFailure : BOOL;
    bBlinking : BOOL;
    bFixed : BOOL;
```

```
    bHorn : BOOL;
END_VAR

VAR
    stInstantDiagnostic : ST_Diagnostic;
    nUi : UDINT;

END_VAR

(* reSet condizioni istantantanee *)
FOR nUi := 0 TO GVL_Diagnostic.cMAXEVENTS -1 DO
    stInstantDiagnostic.aCritical[nUi]  := 0;
    stInstantDiagnostic.aError[nUi]     := 0;
    stInstantDiagnostic.aWarning[nUi]   := 0;
    stInstantDiagnostic.aInfo[nUi]      := 0;
    stInstantDiagnostic.aVerbose[nUi]   := 0;
END_FOR

(* filtri, etc *)

(* imposta condizioni istantanee *)
stInstantDiagnostic.aCritical[0] := bHwError;
stInstantDiagnostic.aCritical[1] := NOT stCtrl.bEmergency;

stInstantDiagnostic.aError[0] := stStatus.eErrorStep <> E_FLM_Step.NoStep;
stInstantDiagnostic.aError[1] := stStatus.stCorking.eErrorStep <>
E_FLM_CorkingStep.NoStep;
stInstantDiagnostic.aError[2] := (stStatus.eStep > E_FLM_Step.Stop) AND
(stIn.fWaterLevel < stParams.fWaterMaxLevel * 0.1);
stInstantDiagnostic.aError[3] := (stStatus.eStep > E_FLM_Step.Stop) AND
(stIn.fMintLevel < stParams.fMintMaxLevel * 0.1);

stInstantDiagnostic.aWarning[0] := (stStatus.eStep > E_FLM_Step.Stop) AND
(stIn.fWaterLevel < stParams.fWaterMaxLevel * 0.3);
stInstantDiagnostic.aWarning[1] := (stStatus.eStep > E_FLM_Step.Stop) AND
(stIn.fMintLevel < stParams.fMintMaxLevel * 0.3);

(* reset condizioni ritenute *)
IF bAck THEN
    FOR nUi := 0 TO GVL_Diagnostic.cMAXEVENTS -1 DO
        stDiagnostic.aCritical[nUi] := 0;
        stDiagnostic.aError[nUi] := 0;
        stDiagnostic.aWarning[nUi] := 0;
        stDiagnostic.aInfo[nUi] := 0;
        stDiagnostic.aVerbose[nUi] := 0;
    END_FOR
    bCritical := FALSE;
    bError := FALSE;
    bWarning := FALSE;
    bInfo := FALSE;
    bVerbose := FALSE;
```

```
    bBlinking := FALSE;
    bHorn := FALSE;
END_IF

(* imposta condizioni ritenute *)
FOR nUi := 0 TO GVL_Diagnostic.cMAXEVENTS -1 DO
    IF stInstantDiagnostic.aCritical[nUi] THEN
        stDiagnostic.aCritical[nUi] := TRUE;
        bCritical := TRUE;
    END_IF
    IF stInstantDiagnostic.aError[nUi] THEN
        stDiagnostic.aError[nUi] := TRUE;
        bError := TRUE;
    END_IF
    IF stInstantDiagnostic.aWarning[nUi] THEN
        stDiagnostic.aWarning[nUi] := TRUE;
        bWarning := TRUE;
    END_IF
    IF stInstantDiagnostic.aInfo[nUi] THEN
        stDiagnostic.aInfo[nUi] := TRUE;
        bInfo := TRUE;
    END_IF
    IF stInstantDiagnostic.aVerbose[nUi] THEN
        stDiagnostic.aVerbose[nUi] := TRUE;
        bVerbose := TRUE;
    END_IF
END_FOR

bFailure := bCritical OR bError;

bBlinking := bFailure;
bHorn := bFailure;
bFixed := bWarning;
```

3.8 Le macchine a stati

FB_FLM_StateMachine è la macchina a stati principale, mentre FB_FLM_CorkingStateMachine è una macchina a stati secondaria.

Ci può essere solo una macchina a stati principale, ma ci possono essere molte macchine a stati secondarie. La macchina a stati principale è quella che risponde ai comandi di start e stop della struttura ST_FLM_Cmd. Le macchine a stati secondarie invece hanno un segnale di start che viene generato dalla macchina a stati principale, mentre il segnale di stop può essere generato dalla macchina a stati principale oppure può essere quello presente nella struttura ST_FLM_Cmd.

La macchina a stati principale legge lo stato della macchina a stati secondaria direttamente dalla struttura ST_FLM_Status.