PumpControl library

This library require other libraries to be installed:

- Utils.sul

Description

This library is to manage pump groups with different types of control

Cascade_d

This function block manages group up to 9 pumps to hold a desired pressure. This control is used with discreet management usually when contactors are used to turn on pumps. Pumps are always turned on to 100% of it's power. basic logic al lowest and highest pressure. If pressure drops below lowest point new pump is added to the group, if pressure raises one pump in a group turns off. It also automatically rotates pumps to even work hours.

Variable	Scope	Type	Description
`Pumps`	IN_OUT	ARRAY[0..9] OF PUMPS_CASCAD_D	An array of pumps to manage
`Start`	INPUT	Bit	Start group
`Reset`	INPUT	Bit	Reset all pump's alarms
`MinPressure`	INPUT	ANY_16	Minimum pressure to add new pump to group
`MaxPressure`	INPUT	ANY_16	Maximum pressure to delete one pump from group
`CurPressure`	INPUT	ANY_16	Current pressure
`TimeTest`	INPUT	ANY_16	How often in seconds to test if new pump should be added or deleted
`MaxTempr`	INPUT	ANY_16	If you have temperature sensor applied to motor or pump to protect from overheat
`CurTempr`	INPUT	ANY_16	Current temperature
`MaxHours`	INPUT	ANY_16	How many hours to work for motor for service and alarm
`FlowDelay`	INPUT	ANY_16	For how many seconds to delay flow check

First you have to declare Pumps array with is an array of 10 `PUMPS_CASCAD_D` objects. The structure of this type is.

Variable	Type	Description
`Enabled`	ANY_16	Only enabled pumps are started

Variable	Type	Description
Start	ANY_16	Started pumps are added to cascade
BlockWork	ANY_16	Block pumps to alarm state
Contactor	ANY_16	Signal that contactor was closed
Flow	ANY_16	Signal from flow relay
Tempr	ANY_16	Temperature from motor or pump
WorkHour	ANY_16	How many hours pump worked
ResetWorkHour	ANY_16	Reset hours pump worked
Alarms	ANY_16	Stores pump alarms

When you declared an array you can call a function block. For instance you have 3 pumps group. The simple setup would be.

```
VAR
    stPumps: ARRAY[0..9] OF PUMP_CASCAD_D;
    fbPumps: Cascade_d;
END_VAR

stPumps[0].Enabled := 1;
stPumps[1].Enabled := 1;
stPumps[2].Enabled := 1;

(* NO signals from contactors  *)
stPumps[0].Contactor := BOOL_TO_INT(X4);
stPumps[1].Contactor := BOOL_TO_INT(X5);
stPumps[2].Contactor := BOOL_TO_INT(X6);

(* NO signals from flow relay  *)
stPumps[0].Flow := BOOL_TO_INT(X7);
stPumps[1].Flow := BOOL_TO_INT(X10);
stPumps[2].Flow := BOOL_TO_INT(X11);

(* Scale pressure measued on AI0 from 0.0 to 16.0 bar *)
AI_P := SCALE(D8030, 0, 4000, 0, 160);
(* Scale Temperature on AI1 from 0.0 to 300.0 C *)
AI_P := SCALE(D8031, 0, 4000, 0, 3000);

fbPump(
    Pumps := stPumps,
    Start := X1,
    Reset := X2,
    MinPressure := 20, (* Equal to 2.0 bar *)
    MaxPressure := 35, (* Equal to 3.5 bar *)
    CurPress := AI_P,
    TimeTest := 10,
    MaxTempr := 400,  (* Equal to 40.0 C *)
```

```
    CurTempr := AI_T,  (* Equal to 40.0 C *)
    MaxHours := 2000,
    FlowDelay := 5
);

Y0 := (stPumps[0].Start = 1);
Y1 := (stPumps[1].Start = 1);
Y2 := (stPumps[2].Start = 1);
```

Note that for `Contactor` and `Flow` and `Enable` properties we set number not Boolean or bit. This is related to the fact that in `FOR` cycle it is impossible to access `.Property` of a bit type in GX Worx 2. thus we have INT types to store BOOL logic.

How to work with alarms

You can also get alarms of any pump like this.

```
IF stPumps[2].Alarms > 0 THEN
    (* there are alarms on this pump *)
END_IF;
```

This way you know that there is an alarm in the pump. If you want to bet exact alarm.

```
IF ISBON(stPumps[2].Alarms, BIT_ALARM_OVERHEAT) THEN
    (* Pump #3 was overheated *)
END_IF;
```

Available alarms are

Alarm Type	Description
`BIT_ALARM_OVERHEAT`	Temperature was too high. Pump is blocked to work until manual reset
`BIT_ALARM_OVERWORK`	Hours set to `MaxHours` reached for this pump. Pump continue working.
`BIT_ALARM_NOFLOW`	There was no water flow detected after `FlowDelay`. Pump is blocked to work until manual reset
`BIT_ALARM_NOCONTACT`	NO of contactor did not return signal. Pump is blocked to work until manual reset. This might happen if you have relay protection of pump and it was triggered.

If you want to integrate with alarms library you can do this.

```
VAR
    fbAMInit: AM_INIT;
    fbAMSet: AM_SET;
```

```
END_VAR

IF M8002 THEN
    (* Pump 1 Overheat *)
    fbAMINIT(iNum := 0, iProcess := 1, iSeverity := 2,
        xLock := FALSE, xLatch := TRUE, xBuzzer := TRUE);
    (* Pump 2 Overheat *)
    fbAMINIT(iNum := 1, iProcess := 1, iSeverity := 2,
        xLock := FALSE, xLatch := TRUE, xBuzzer := TRUE);

    (* Pump 1 Flow *)
    fbAMINIT(iNum := 2, iProcess := 1, iSeverity := 2,
        xLock := FALSE, xLatch := TRUE, xBuzzer := TRUE);
    (* Pump 2 Flow *)
    fbAMINIT(iNum := 3, iProcess := 1, iSeverity := 2,
        xLock := FALSE, xLatch := TRUE, xBuzzer := TRUE);
END_IF

fbAmSet(iNum := 0, xState := ISBON(stPumps[0].Alarms, BIT_ALARM_OVERHEAT));
fbAmSet(iNum := 1, xState := ISBON(stPumps[1].Alarms, BIT_ALARM_OVERHEAT));
fbAmSet(iNum := 2, xState := ISBON(stPumps[0].Alarms, BIT_ALARM_NOFLOW));
fbAmSet(iNum := 3, xState := ISBON(stPumps[1].Alarms, BIT_ALARM_NOFLOW));
```