FUNCTION BLOCK FB_GenerateP5

To be completed

Au moment du Start, le temps du polynôme commence à incrementer.

I/O	Type	Name	Comment
VAR_INPUT			
	BOOL	Enable	
	BOOL	bStartForward	Start of motion at rising edge
	LREAL	startPosition_mm	Start position of motion
	LREAL	endPosition_mm	End position of motion
	UDINT	udiCyclTime_ms	Cycle time in ms of the task where FB is running
	UDINT	Cycle Time Multiplier, default is 1000 for 1000 ms	

FUNCTION_BLOCK FB_GenerateP5 VAR_INPUT // Start of motion at rising edge. bStartForward : BOOL; // Should be 0 for first version, could be extended for any position startPosition_mm : LREAL := 0; // Should be limited to 100 mm endPosition_mm : LREAL := 0; // Cycle time in ms of the task where FB is running. udiCyclTime_ms : UDINT; // Cycle time multiplier, used to compute duration of motion // This could be made automatically later, // Now result in Output // Cycle Time Multiplier, default is 1000 for 1000 ms udiCycleTimeMult : UDINT; END_VAR VAR_IN_OUT // Used for verification of data, we do not nead them for motion arGoToPos : ARRAY[0..GVL_CyclSetPoint.c_uiMaxProfilePoints] OF LREAL; END_VAR VAR_OUTPUT // Is time to execute the profile in ms // Multiplication of udiCyclTime_ms x udiCycleTimeMult udiMotionTime_ms : UDINT; // Motion finished bDoneForward : BOOL; // Motion set position lrSetPosition : LREAL; // True if udiCycleTimeMult Exceeds, FB not executed bErrorTimeMultToBig : BOOL; // Point generator active bActive : BOOL; END_VAR VAR // Time index of position discreteTimeIndex : UDINT; // rising edge of start rStartForward : R_TRIG; // Ce machin là va devoir être expliqué avec pas mal de détail en théorie. lrTimeScaler : LREAL; END_VAR

Header at 2024 05 29

```
FUNCTION_BLOCK FB_GenerateP5
VAR_INPUT
    // Std Inout for EnableDoneBase
    Enable              : BOOL;
    // Start of motion at rising edge.
    StartForward       : BOOL;
    // Should be 0 for first version, could be extended for any position
    startPosition_mm    : LREAL := 0;
    // Should be limited to 100 mm
    endPosition_mm      : LREAL := 0;
    // Cycle time in ms of the task where FB is running.
    udiCyclTime_ms      : UDINT;
    // Cycle time multiplier, used to compute duration of motion
    // This could be made automatically later,
    // Now result in Output
    // Cycle Time Multiplier, default is 1000 for 1000 ms
    udiCycleTimeMult    : UDINT;
END_VAR
VAR_IN_OUT
    // Used for verification of data, we do not nead them for motion
    arGoToPos           : ARRAY[0..GVL_CyclSetPoint.c_uiMaxProfilePoints] OF
LREAL;
END_VAR
VAR_OUTPUT
    // Motion finished
    Done                : BOOL;
    // Point generator active
    Active              : BOOL;
    Error               : BOOL;
    // Is time to execute the profile in ms
    // Multiplication of udiCyclTime_ms x udiCycleTimeMult
    udiMotionTime_ms    :  UDINT;
    // Motion set position
    lrSetPosition       : LREAL;
END_VAR
VAR
    // True if udiCycleTimeMult Exceeds, FB not executed
    bErrorTimeMultToBig : BOOL;
    // Time index of position
    discreteTimeIndex   : UDINT;
    // Ce machin là va devoir être expliqué avec pas mal de détail en théorie.
    lrTimeScaler        : LREAL;
    //   Internal state machine
    eInOpeDoneBase      : E_InOperationDoneBase;
    eSubActiveP5        : E_SubActiveP5;
END_VAR
```

Code # 2024 05 29

```
CASE eInOpeDoneBase OF
    E_InOperationDoneBase.Idle  :
        IF Enable THEN
            eInOpeDoneBase := E_InOperationDoneBase.Init;
        END_IF

    E_InOperationDoneBase.Init  :
        IF udiCycleTimeMult > GVL_CyclSetPoint.c_uiMaxProfilePoints THEN
            eInOpeDoneBase := E_InOperationDoneBase.Error;
        ELSIF Enable THEN
```

```
            // Hence this value cannot be modified while moving
            udiMotionTime_ms := udiCyclTime_ms * udiCycleTimeMult;
            eInOpeDoneBase := E_InOperationDoneBase.InOp;
            eSubActiveP5 := E_SubActiveP5.StartPointOut;
        ELSE
            eInOpeDoneBase := E_InOperationDoneBase.Idle;
        END_IF

    E_InOperationDoneBase.InOp  :
        // No error condition
        IF Enable THEN
            CASE eSubActiveP5 OF
                E_SubActiveP5.StartPointOut :
                    IF StartForward THEN
                        lrSetPosition := startPosition_mm;
                        discreteTimeIndex := 0;
                        lrTimeScaler := 1/UDINT_TO_LREAL(udiCycleTimeMult);
                        eSubActiveP5 := E_SubActiveP5.Running;
                    END_IF

                E_SubActiveP5.Running       :
                    // Matlab operation with some modifications :
                    // position_yAxis(iLoop) = (t_xAxis(iLoop)^3 * P5_position(4)
+ t_xAxis(iLoop)^4 * P5_position(5) + t_xAxis(iLoop)^5 * P5_position(6)) *
y_scaling_mm;
                    arGoToPos[discreteTimeIndex] := startPosition_mm
+

(EXPT(lrTimeScaler*discreteTimeIndex,3) * GVL_CyclSetPoint.P5[4] +

EXPT(lrTimeScaler*discreteTimeIndex,4) * GVL_CyclSetPoint.P5[5] +

EXPT(lrTimeScaler*discreteTimeIndex,5) * GVL_CyclSetPoint.P5[6]) * (endPosition_mm
- startPosition_mm);
                    // Set new value
                    lrSetPosition := arGoToPos[discreteTimeIndex];
                    // inc at each cycle
                    discreteTimeIndex := discreteTimeIndex + 1;
                    IF discreteTimeIndex > udiCycleTimeMult THEN
                        eSubActiveP5 := E_SubActiveP5.EndOfProfile;
                    END_IF // discreteTimeIndex > udiCycleTimeMult

                    E_SubActiveP5.EndOfProfile  :
                        ;

                    IF eSubActiveP5 = E_SubActiveP5.EndOfProfile THEN
                        eInOpeDoneBase := E_InOperationDoneBase.Done;
                    END_IF
            END_CASE
        ELSE
            eInOpeDoneBase := E_InOperationDoneBase.Idle;
        END_IF // Enable

    E_InOperationDoneBase.Done  :
        IF NOT StartForward THEN
            eInOpeDoneBase := E_InOperationDoneBase.InOp;
            // reset internal state machine
            eSubActiveP5 := E_SubActiveP5.StartPointOut;
        END_IF

    E_InOperationDoneBase.Error :
        bErrorTimeMultToBig := TRUE;
        IF NOT Enable THEN
            bErrorTimeMultToBig := FALSE;
        END_IF

END_CASE

(*
    Manage outputs
*)
Active := (eInOpeDoneBase = E_InOperationDoneBase.InOp);
Done := (eInOpeDoneBase = E_InOperationDoneBase.Done);
Error := (eInOpeDoneBase = E_InOperationDoneBase.Error);
```