# reACTION 模块通过位置插补方法在喷码触发中的应用

## 1、需求说明

某客户有一 15 色柔凹组合印刷机，前面 2 色凹版，中间 8 色柔版，后面 3 色凹版。产品印刷工艺为所有凹版印刷的为局部透明光油，套色电眼无法检测且后面柔版印刷需要与前面的凹版印刷的透明光油套色，因此在第 1 色凹版处安装一个喷码仪，1 色版辊每次转到某一相位时触发喷码仪喷一个色标，后面的印刷色组使用单标记方式与该色标套色。

## 2、系统配置

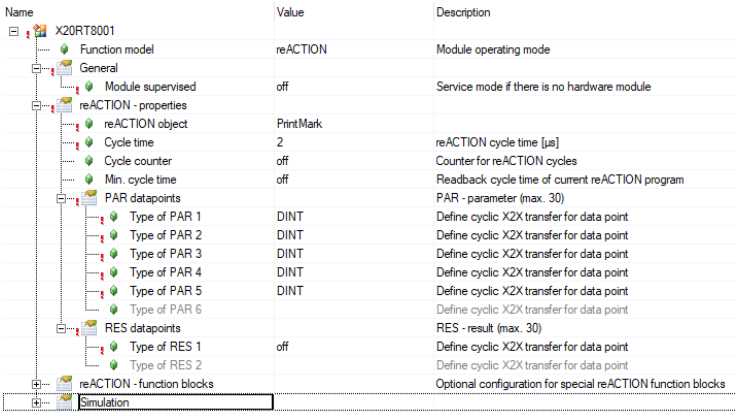
PPC2100

X20RT8001

## 3、实现方式

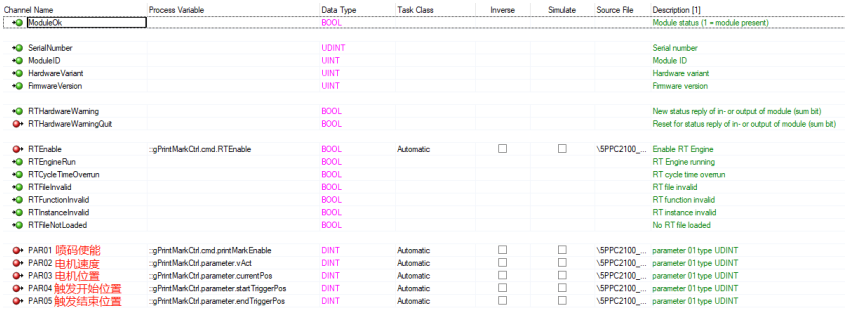
版辊电机设定位置 113，每个任务周期 1.2ms 更新一次给 reACTION 程序接口，reACTION 循环周期设定为 2μs，reACTION 内部当版辊电机设定位置更新时使用更新的版辊位置，不更新时 reACTION 程序内部每 2μs 根据电机上次更新的位置及速度插补生成实时位置，根据 reACTION 程序内部每 2μs 更新的电机位置达到触发喷码的相位范围时将输出信号置 1。

### 3.1 X20RT8001 硬件配置



| Name | Value | Description |
|---|---|---|
| X20RT8001 | | |
| Function model | reACTION | Module operating mode |
| General | | |
| Module supervised | off | Service mode if there is no hardware module |
| reACTION - properties | | |
| reACTION object | PrintMark | |
| Cycle time | 2 | reACTION cycle time [µs] |
| Cycle counter | off | Counter for reACTION cycles |
| Min. cycle time | off | Readback cycle time of current reACTION program |
| PAR datapoints | | PAR - parameter (max. 30) |
| Type of PAR 1 | DINT | Define cyclic X2X transfer for data point |
| Type of PAR 2 | DINT | Define cyclic X2X transfer for data point |
| Type of PAR 3 | DINT | Define cyclic X2X transfer for data point |
| Type of PAR 4 | DINT | Define cyclic X2X transfer for data point |
| Type of PAR 5 | DINT | Define cyclic X2X transfer for data point |
| Type of PAR 6 | | Define cyclic X2X transfer for data point |
| RES datapoints | | RES - result (max. 30) |
| Type of RES 1 | off | Define cyclic X2X transfer for data point |
| Type of RES 2 | | Define cyclic X2X transfer for data point |
| reACTION - function blocks | | Optional configuration for special reACTION function blocks |
| Simulation | | |

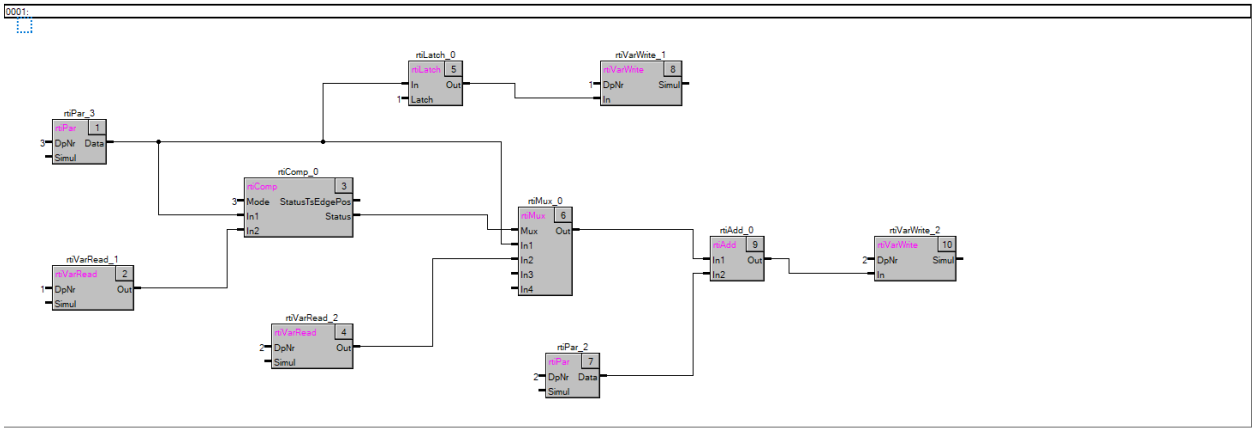
### 3.2 X20RT8001 IO 映射



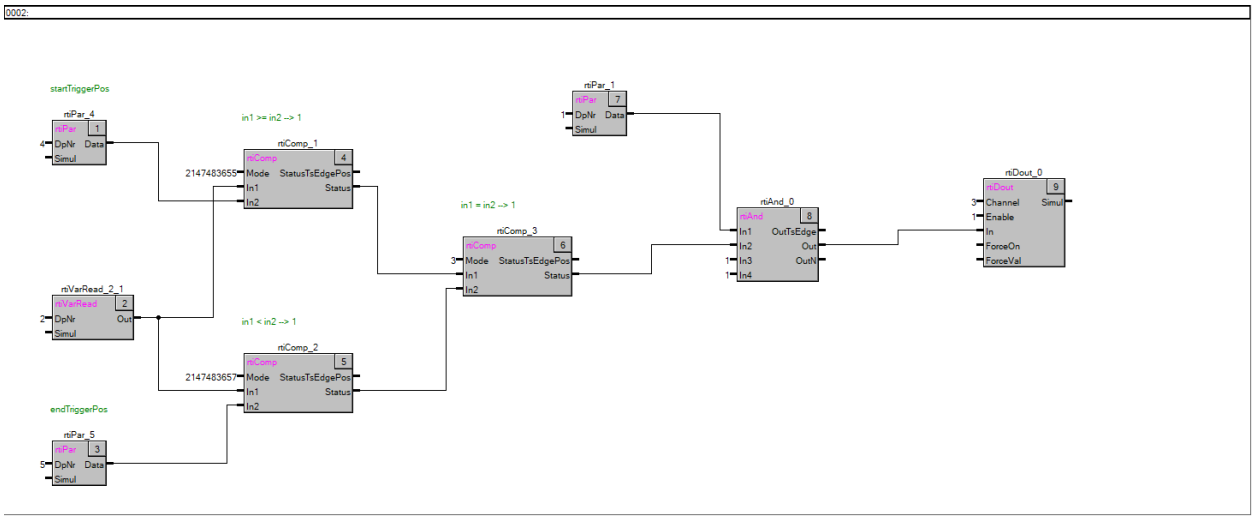
| Channel Name | Process Variable | Data Type | Task Class | Inverse | Simulate | Source File | Description [1] |
|---|---|---|---|---|---|---|---|
| ModuleOk | | BOOL | | | | | Module status (1 = module present) |
| SerialNumber | | UDINT | | | | | Serial number |
| ModuleID | | UINT | | | | | Module ID |
| HardwareVariant | | UINT | | | | | Hardware variant |
| FirmwareVersion | | UINT | | | | | Firmware version |
| RTHardwareWarning | | BOOL | | | | | New status reply of in- or output of module (sum bit) |
| RTHardwareWarningQuit | | BOOL | | | | | Reset for status reply of in- or output of module (sum bit) |
| RTEnable | ::gPrintMarkCtrl.cmd.RTEnable | BOOL | Automatic | ☐ | ☐ | \$PPC2100_... | Enable RT Engine |
| RTEngineRun | | BOOL | | | | | RT Engine running |
| RTCycleTimeOverrun | | BOOL | | | | | RT cycle time overrun |
| RTFileInvalid | | BOOL | | | | | RT file invalid |
| RTFunctionInvalid | | BOOL | | | | | RT function invalid |
| RTInstanceInvalid | | BOOL | | | | | RT instance invalid |
| RTFileNotLoaded | | BOOL | | | | | No RT file loaded |
| PAR01 喷码使能 | ::gPrintMarkCtrl.cmd.printMarkEnable | DINT | Automatic | ☐ | ☐ | \$PPC2100_... | parameter 01 type UDINT |
| PAR02 电机速度 | ::gPrintMarkCtrl.parameter.vAct | DINT | Automatic | ☐ | ☐ | \$PPC2100_... | parameter 01 type UDINT |
| PAR03 电机位置 | ::gPrintMarkCtrl.parameter.currentPos | DINT | Automatic | ☐ | ☐ | \$PPC2100_... | parameter 01 type UDINT |
| PAR04 触发开始位置 | ::gPrintMarkCtrl.parameter.startTriggerPos | DINT | Automatic | ☐ | ☐ | \$PPC2100_... | parameter 01 type UDINT |
| PAR05 触发结束位置 | ::gPrintMarkCtrl.parameter.endTriggerPos | DINT | Automatic | ☐ | ☐ | \$PPC2100_... | parameter 01 type UDINT |

### 3.3 reACTION 程序实现

位置插补程序：




设定相位范围信号触发程序：




## 3.4 C 语言接口程序实现

由于 reACTION 程序任务周期为 2 μ s，需将 C 语言程序中的速度单位 μ m/s 转换为 nm/2 μ s。为确保高低速最终喷码位置一致，将喷码仪的延迟时间导致的速度变化引起的喷码位置偏移叠加到喷码触发开始位置及喷码触发结束位置。

```
    /*************************************************************/
    /*************************喷色标控制**************************/
    /*************************************************************/
    //1、开始喷印
    //2、当前速度
    //3、当前相位
    //4、喷印开始相位
    //5、喷印结束相位

    gPrintMarkCtrl.cmd.RTEnable = 1;                                    //reAction功能开启
    gPrintMarkCtrl.parameter.reActionCycTime = 2;                       //reAction周期us
    gPrintMarkCtrl.parameter.homingMode      = 0;                       //0:寻参一次;1:每次寻参
    gPrintMarkCtrl.parameter.pruIndex        = gRcpCtrl.general.printMarkIndex;     //色标触发色组
    gPrintMarkCtrl.parameter.triggerMode     = gRcpCtrl.general.printMarkMode;      //0:固定宽度;1:固定时间
    gPrintMarkCtrl.parameter.triggerWidth    = gRcpCtrl.general.printMarkWidth;     //触发宽度um
    gPrintMarkCtrl.parameter.triggerTime     = gRcpCtrl.general.printMarkTime;      //触发时间ms
    gPrintMarkCtrl.parameter.printMarkPos    = gRcpCtrl.general.printMarkPos;       //喷色标设定位置um
    gPrintMarkCtrl.parameter.compTime        = gRcpCtrl.general.printMarkCompTime;  //补偿时间us
```

```

    //开始喷印
    TON_PrintMarkEn.IN = (gActData.general.machineLine == 1 && gHMI.printUnit.diHipDown[gPrintMarkCtrl.parameter.pruIndex] && gHMI.printUnit.diModeSelect[gPrintMarkCtrl.parameter.pruIndex]);
    TON_PrintMarkEn.PT = gRcpCtrl.time.nipDownlBylDelay;
    TON(&TON_PrintMarkEn);
    gPrintMarkCtrl.cmd.printMarkEnable = TON_PrintMarkEn.Q;

    //当前速度;单位nm/2us
    gPrintMarkCtrl.parameter.vAct = (REAL)gActData.printUnit.vAct[gPrintMarkCtrl.parameter.pruIndex] * (REAL)gPrintMarkCtrl.parameter.reActionCycTime * 0.001 + 0.5;

    //当前加速度;单位nm/2us2
    cycUpdateTimer[gPrintMarkCtrl.parameter.pruIndex] += rcycT;
    if(gActData.printUnit.cycUpdate[gPrintMarkCtrl.parameter.pruIndex] == 1)
    {
        gActData.printUnit.cycUpdate[gPrintMarkCtrl.parameter.pruIndex] = 0;

        //单位um/s2
        gActData.printUnit.aAct[gPrintMarkCtrl.parameter.pruIndex] = (REAL)(gActData.printUnit.vAct[gPrintMarkCtrl.parameter.pruIndex] - vActOld[gPrintMarkCtrl.parameter.pruIndex])/cycUpdateTimer[gPrintMarkCtrl.parameter.pruIndex]*1000.0;
        //单位nm/2us2
        gPrintMarkCtrl.parameter.aAct = 0;
        gPrintMarkCtrl.parameter.aActHalf = 0;

        vActOld[gPrintMarkCtrl.parameter.pruIndex] = gActData.printUnit.vAct[gPrintMarkCtrl.parameter.pruIndex];
        cycUpdateTimer[gPrintMarkCtrl.parameter.pruIndex] = 0;
    }
```

```
    //当前相位,单位nm
    if(sImageOld != gRcpUser.general.sImage)
    {
        for(pruIndex = 0;pruIndex< gRcpFix.nbPRU;pruIndex++)
        {
            gActData.printUnit.triggerHomingOK[pruIndex] = 0;
        }
    }
    sImageOld = gRcpUser.general.sImage;

    for(pruIndex = 0;pruIndex< gRcpFix.nbPRU;pruIndex++)
    {
        if(gPrintMarkCtrl.parameter.homingMode == 1 && gActData.general.machineLine == 0)
        {
            gActData.printUnit.triggerHomingOK[pruIndex] = 0;
        }

        if(gActData.printUnit.triggerHomingOK[pruIndex] == 0 && gActData.general.machineLine == 1 && gHMI.printUnit.diModeSelect[pruIndex] == 1)
        {
            if(gActData.printUnit.sActTriggerPosEdge[pruIndex] != sActTriggerPosEdgeOld[pruIndex])
            {
                gActData.printUnit.triggerHomingOK[pruIndex] = 1;
                gActData.printUnit.sHomingTrigger[pruIndex] = gActData.printUnit.sActTriggerPosEdge[pruIndex];
            }
        }
        sActTriggerPosEdgeOld[pruIndex] = gActData.printUnit.sActTriggerPosEdge[pruIndex];

        //计算版辊实时相位
        sHomingTemp[pruIndex] = gActData.printUnit.sAct[pruIndex] - gActData.printUnit.sHomingTrigger[pruIndex];
        if(sHomingTemp[pruIndex] >= gRcpUser.general.sImage)
        {
            gActData.printUnit.sHomingTrigger[pruIndex] = gActData.printUnit.sHomingTrigger[pruIndex] + gRcpUser.general.sImage;
        }
        else if(sHomingTemp[pruIndex] < 0)
        {
            gActData.printUnit.sHomingTrigger[pruIndex] = gActData.printUnit.sHomingTrigger[pruIndex] - gRcpUser.general.sImage;
        }
        gActData.printUnit.sOffsetTrigger[pruIndex] = gActData.printUnit.sAct[pruIndex] - gActData.printUnit.sHomingTrigger[pruIndex];
    }

    gPrintMarkCtrl.parameter.currentPos = gActData.printUnit.sOffsetTrigger[gPrintMarkCtrl.parameter.pruIndex]*1000;
```

```
    //喷色标开始相位-结束相位,单位nm
    gPrintMarkCtrl.parameter.startTriggerPos = gPrintMarkCtrl.parameter.printMarkPos * 1000;
    if(gPrintMarkCtrl.parameter.triggerMode == 0)
    {
        gPrintMarkCtrl.parameter.endTriggerPos = gPrintMarkCtrl.parameter.startTriggerPos + gPrintMarkCtrl.parameter.triggerWidth * 1000;
    }
    else
    {
        gPrintMarkCtrl.parameter.endTriggerPos = gPrintMarkCtrl.parameter.startTriggerPos + (gPrintMarkCtrl.parameter.triggerTime*gActData.printUnit.vAct[gPrintMarkCtrl.parameter.pruIndex]);

        gPrintMarkCtrl.parameter.endTriggerPos = gPrintMarkCtrl.parameter.endTriggerPos + 0.5 * 0.001 * gActData.printUnit.aAct[gPrintMarkCtrl.parameter.pruIndex]*gPrintMarkCtrl.parameter.triggerTime*gPrintMarkCtrl.parameter.triggerTime;
    }

    //补偿时间
    gPrintMarkCtrl.parameter.compTimeS = (REAL)gPrintMarkCtrl.parameter.compTime * (REAL)gActData.printUnit.vAct[gPrintMarkCtrl.parameter.pruIndex] * 0.001;
    gPrintMarkCtrl.parameter.startTriggerPos = gPrintMarkCtrl.parameter.startTriggerPos - gPrintMarkCtrl.parameter.compTimeS;
    gPrintMarkCtrl.parameter.endTriggerPos   = gPrintMarkCtrl.parameter.endTriggerPos - gPrintMarkCtrl.parameter.compTimeS;


    nextCycPos = gPrintMarkCtrl.parameter.currentPos + (rcycT * gActData.printUnit.vAct[gPrintMarkCtrl.parameter.pruIndex] * 0.001) + (0.5 * 0.000001 * gActData.printUnit.aAct[gPrintMarkCtrl.parameter.pruIndex] * rcycT * rcycT);
    gPrintMarkCtrl.status.cycCount = gPrintMarkCtrl.status.cycCount;
```