ČVUT Fakulta Strojní

Řízení Programovatelnými Automaty

Dokumentace pro úlohu "Opískování"

Kirill Rassudikhin

Kirill.Rassudikhin@fs.cvut.cz

Kapitola 1

Zadání

Zadáním je vytvořit řidicí program pro jednoduchou technologickou operaci. V mém případě je to opiskování.

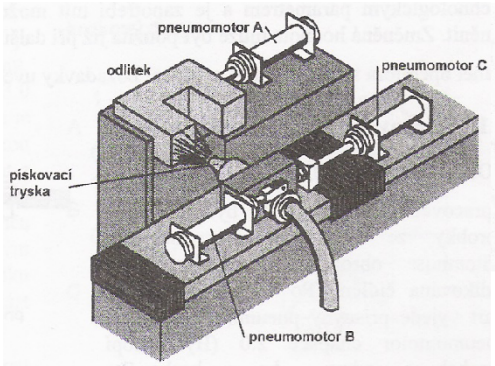
Slovní zadání:

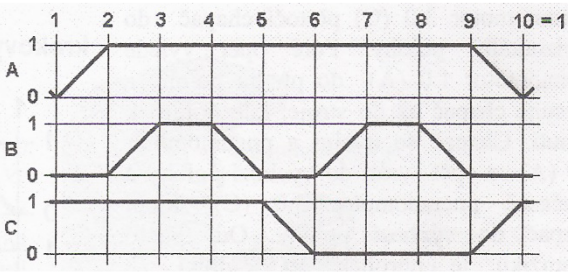
Úloha A – Přípravek pro opískování odlitku

Požadovaná funkce

Na odlitku je třeba opískovat dvě ramena. Odlitek je vložen ručně do upínacího přípravku a tlačítkem START je vydán pokyn pro začátek operace. Obrobek je upnut pneumotorem A. Potom na předem nastavenou dobu Ta otevře pneumotor B ventil pískovací trysky. Tato doba je parametrem, který je možno pro následující opracovávaný odlitek změnit z operátorského pracoviště – klávesnice PC. Po opětném uzavření ventilu trysky přesune pneumotor C trysku ke druhému ramenu obrobku. Operace opískování se opakuje se stejnou dobou trvání. Po skončení druhého opískování se vrátí pneumotor C do výchozí polohy. Pneumotor A uvolní odlitek, který může být ručně vyjmut z přípravku.

Náčrt situace a krokový diagram:





Kapitola 2

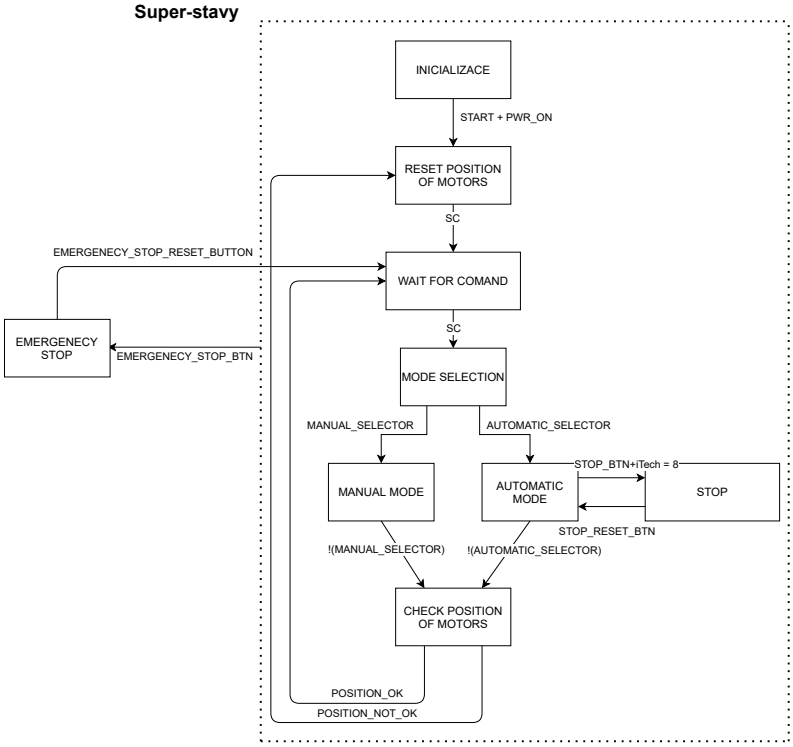
Analýza

Analýza zadání je sestavení stavového diagramu pro stavový automat, který bude řídit pracoviště pro moji technologickou operaci.

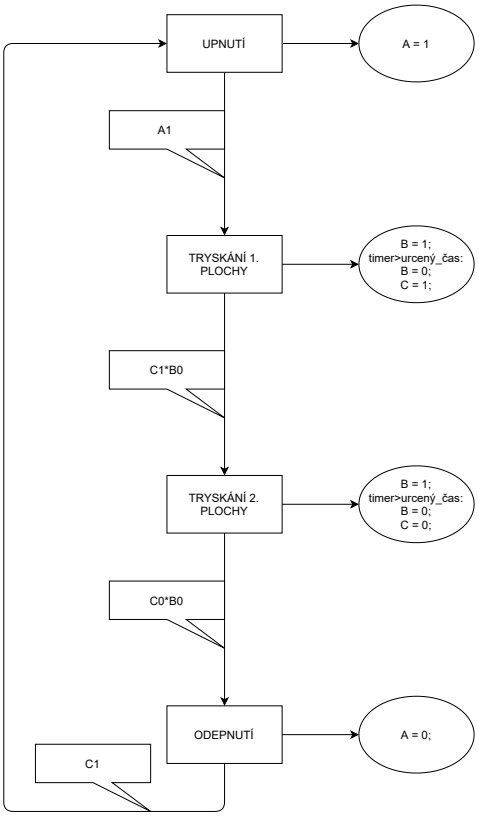
Stavový diagram se zkladá ze třech stavových diagramů. Důvodem je standart PackML, který rozděluje stavový automat na Technologickou a Obecnou části. První část je "Superstavy", je obecná část, popisuje netechnologickou rutinu jako inicializace, výběr modu a t.d. "Automatický mod"je stavový diagram pro, překvapivě, automatický mod, čili samotnou technologickou sekvenci "Opiskování". "Manualní mod"popistuje chování zařízení v manuálním režímu, který umožňuje pracovnikovi udělat každou možnou akci na pracoviště ručně pomocí tlačítek (třeba zvlášť vyjet pneumotorem A). Manualní mod je zatím ještě neimplementovan do prográmu.

Stavové diagramy pro moji sekvenci inspirované standartem PackML jsou úvedené na příští strance.

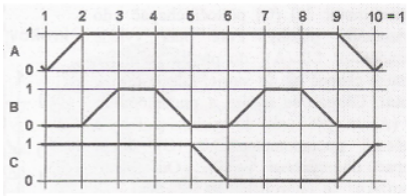
Super-stavy



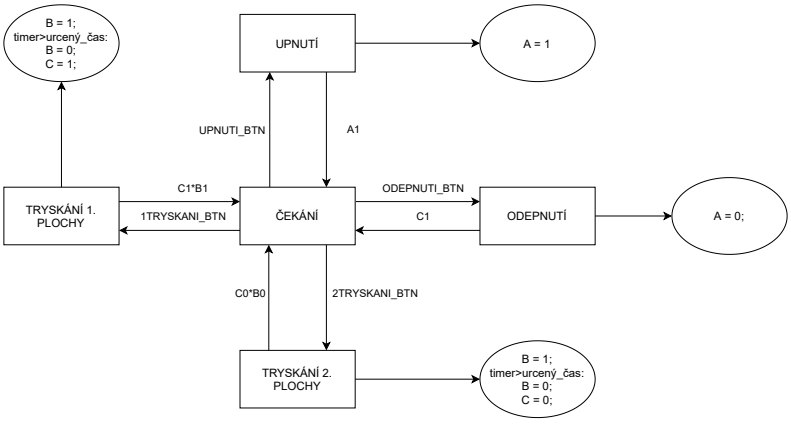
Automatický mod



Krokový diagram



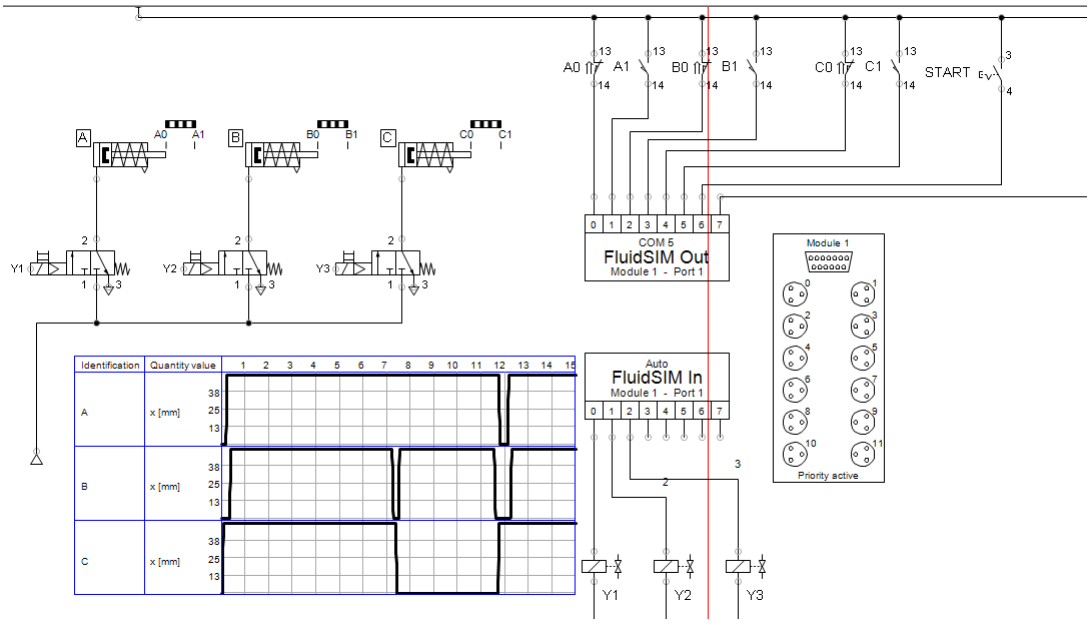
Manualní mod



Kapitola 3

Návrh polní úrovně

Jako polní instrumentaci jsem si zvolil tři jednočinné válce s monostabilními rozváděči. Jako důkaz fungování sekvence je taky úveden diagram polohy pístů pro válce A,B,C.



Kapitola 4

Výběr PLC

Pro danou úlohu jsme používali velmi dostupný PLC UniPi Neuron M203, který je postaven na Raspberry Pi. Důvodem pro jeho použití je taky možnost použití opensoursových softwarů a firmawarů pro jeho programování a obsluhu. Má dostatečný počet výstupu a vstupů pro zápojení technologické sekvence, ale pokud bych chtěl k tomu ještě zapojit přímo každé tlačitko, úvedené v analýze zadání, ty vystupy by mi už nestačili. Nicmeně poměr cena/kvalita u UniPi Neuron je dobrý, takže pokud bych měl použit levné a dostupné řešení, použil bych taky UniPi Neuron M203.

Má port pro Ethernet, RS485, Analogové, Digitální vstupy/výstupy a Releové výstupy. Uplný seznam Vlastností je úveden dole.

Basic features

Digital inputs	**20**	**Common features**
Digital outputs	**4**	Raspberry Pi 3 Model B+ computer
Relay outputs	**14**	(1.4GHz quad-core CPU, 1GB RAM)
Analog inputs	**1**	4× USB 2.0
Analog outputs	**1**	1× 300Mbps Ethernet
RS485 serial interface	**1**	Wi-Fi, Bluetooth,
1-Wire bus	**1**	Aluminium chassis (IP20)

Obrázek 4.1: Uplný seznam Vlastností UniPi Neuron M203

Pokud bych měl zvolit více komerční řešení pro tento projekt. Použil bych produkce s kterou už jsem pracoval. Proto bych si zvolil buď SIEMENS SIMATIC S7-1200 anebo standartní plc od BR AUTOMATION: X20CP1586.

PRODUKTOVÉ ČÍSLO:

X20CP1586

POPIS:

- Intel Atom 1.6 GHz processor with an additional I/O processor
- Onboard Ethernet, POWERLINK with poll response chaining and USB
- 1 slot for modular interface expansion
- CompactFlash as removable application memory
- 512 MB DDR2 SDRAM
- Fanless

Obrázek 4.2: X20CP1586

Obrázek 4.3: SIEMENS SIMATIC S7-1200

Kapitola 5

Seznam I/O

Název	Adresa	Funkce
I Start	IX1.0	Start button
I PowerOn	IX1.1	Power on button
I StopBtn	IX1.2	Stop button
I StopResBtn	IX1.3	Stop reset button
I EmStopBtn	IX1.4	Emergency stop button
I EmStopResBtn	IX1.5	Emergency stop reset button
I ManualSelector	IX1.6	Manual mode selector button
I AutomaticSelector	IX1.7	Automatic mode selector button
I A0	IX0.0	End sensor from A motor
I A1	IX0.1	End sensor from A motor
I B0	IX0.2	End sensor from B motor
I B1	IX0.3	End sensor from B motor
I C0	IX0.4	End sensor from C motor
I C1	IX0.5	End sensor from C motor

Tabulka 5.1: Seznam vstupů

Název	Adresa	Funkce
Q A	QX0.4	A motor control
Q B	QX0.5	B motor control
Q C	QX0.6	C motor control

Tabulka 5.2: Seznam výstupů

Kapitola 6

Algoritmus

Obecný stavový automat pro "Super-stavy"funguje na příkazu CASE IF. CASE IF je nástroj pro programování stavového automatu v jazyce ST. Uvnitř CASE IF napišu jednotlivé stavy a přechody mezi nimi. Když je ten kod výkonaván cyklicky, vykoná se jenom jeden příslušný podmínkám (vstupům) stav.

Technologická sekvence funguje pomocí principu "maskování". To znamena že každý vstup je zapsan "maskován"do jednotlivého bitů proměnné typu byte, do takzvaného vstupního slova. Takže existuje matice přechodu z jedného stavu do přištího a matice výstupu, do které je obdobným způsobém, jako do vstupního slova jsou zapsáné výstupy pro každý jednotlivý stav. Ten princip funguje tak, že pokud vstupní slovo je totožné s řádkém matice přechodu - vykonavá se řádek matice výstupu, čili mapuji se řádky z matice výstupu na fyzické výstupy.

Kapitola 7

Zdrojový kód

```
IF (I_EMSTOPBTN = true) THEN

    SuperStates := ST_EMERGENCY_STOP;

END_IF;


CASE SuperStates OF

  ST_INICIALIZACE:
    SuperSequenceState := 1;
    (*IF (I_Start = true) AND (I_PowerOn = true) THEN*) (*Uprava 06.05.2021*)

      SuperStates := ST_RESET_POSITION_OF_MOTORS;

    (*END_IF;*)

  ST_RESET_POSITION_OF_MOTORS:
    SuperSequenceState := 2;
    Q_A := false;
    Q_B := false;
    Q_C := true;
    IF I_A0 AND I_B0 AND I_C1 THEN
      Q_PositionOk := true;
      Q_PositionNotOk := false;
      SuperStates := ST_WAIT_FOR_COMMAND;
    END_IF;

  ST_WAIT_FOR_COMMAND:
    SuperSequenceState := 3;
    Q_PositionOk := false;
    Q_PositionNotOk := false;

    SuperStates := ST_MODE_SELECTION;

  ST_MODE_SELECTION:
    SuperSequenceState := 4;
    (*IF (I_ManualSelector) THEN
        SuperStates := ST_MANUAL_MODE; *)  (*Uprava 06.05.2021*)
    (*ELSIF (I_AutomaticSelector) THEN *)
        SuperStates := ST_AUTOMATIC_MODE;
    (*END_IF;*)    (*Uprava 06.05.2021*)
```

```
    (*TODO// Comeback from this state to STOP*)

  ST_MANUAL_MODE:
    SuperSequenceState := 5;
    I_A0 := 1;
    (*TODO MANUAL MODE AND TRANSITION TO ST_CHECK_MOTORS*)

  ST_AUTOMATIC_MODE:
    SuperSequenceState := 6;

    (*  Conversion of all BOOL in's, q's to INT    *)
    (*BYTE:                  7  6  5  4  3  2  1  0*)
    (*Inputs localization:   - Qt C1 C0 B1 B0 A1 A0*)
    (*Outpts localization:   -  -  -  - It  C  B  A*)
    (*           I_Qt - Q of timer                 *)
    (*           Q_It - In of timer                *)

    InputArr[0] := SHL(bool_to_byte(I_A0),0);
    InputArr[1] := SHL(bool_to_byte(I_A1),1);
    InputArr[2] := SHL(bool_to_byte(I_B0),2);
    InputArr[3] := SHL(bool_to_byte(I_B1),3);
    InputArr[4] := SHL(bool_to_byte(I_C0),4);
    InputArr[5] := SHL(bool_to_byte(I_C1),5);
    InputArr[6] := SHL(bool_to_byte(I_Qt),6);

    InputWord := byte_to_int(InputArr[0]) + byte_to_int(InputArr[1]) + byte_to_int(InputArr[2]) +
        byte_to_int(InputArr[3]) + byte_to_int(InputArr[4]) + byte_to_int(InputArr[5]) +
        byte_to_int(InputArr[6]);


    (*   End of Conversion   *)


    IF (InputWord = TRANS_MATX[iTech]) THEN
      (*Making the outputs HIGH/LOW by going throught bitwise*)
      FOR iTechBit:= 0 TO 7 DO
          (*Going throught one row of OUTPT_MATX and checking, which outputs should be TRUE*)
          IF ((int_to_byte(OUTPT_MATX[iTech]) AND SHL(ByteCompare, iTechBit)) <> 0) THEN
            Qs[iTechBit] := true;
          ELSE
            Qs[iTechBit] := false;
          END_IF;
      END_FOR;

      (*Mapping of outputs*)
      Q_A := Qs[0];
      Q_B := Qs[1];
      Q_C := Qs[2];
      Q_It := Qs[3];
      (*End of mapping of outputs*)

      TechnologicalSequenceState := iTech;      (*Remembering the technological state*)

      iTech := iTech + 1;                       (*Going to the next state*)

    END_IF;

    (*Actions after on technological sequence is done*)
```

```
    (*Program has gone throught all technological states*)
    IF (iTech > 8) THEN

      iTech := 0;                                (*Reset sequence*)
    END_IF;

   (*ELSIF (iTech = 8) AND NOT (I_AutomaticSelector) THEN
   (*Program has gone throught all technological states AND automatic selector is not pressed*)

    (*  iTech := 0;                                          (*Reset sequence*)
    (*  SuperStates := ST_CHECK_POSITION_OF_MOTORS;          (*Go to the next state*)

    (*ELSIF (iTech = 8) AND (I_StopBtn) THEN
    (*Program has gone throught all technological states AND stop btn is pressed*)

     (* SuperStates := ST_STOP;                                (*Go to stop state*)

   (* END_IF;*)


  ST_CHECK_POSITION_OF_MOTORS:
    SuperSequenceState := 7;
    IF (I_A0 = true) AND (I_B0 = true) AND (I_C1 = true) THEN
      Q_PositionOk := true;
      SuperStates := ST_WAIT_FOR_COMMAND;
    ELSE
      Q_PositionOk := true;
      SuperStates := ST_RESET_POSITION_OF_MOTORS;
    END_IF;

  ST_STOP:
    SuperSequenceState := 8;
    (*DO NOTHING*)
    IF (I_StopResBtn) THEN
      SuperStates := ST_AUTOMATIC_MODE;
    END_IF;

  ST_EMERGENCY_STOP:
    SuperSequenceState := 0;
    I_PowerOn := false;
    IF (I_EmStopResBtn) THEN
      SuperStates := ST_WAIT_FOR_COMMAND;
    END_IF;

END_CASE;



(*Function block calling*)

TON_0(IN := I_B1, PT := BlastingTime, Q => I_Qt);


(*End of Function block calling*)
```