Laborator 5

Timere și semnale I/O digitale

Drd. ing. Mihai Pârlea
Drd. ing. Andrei Roșu
As. drd. ing. Alex Dumitrache
As. drd. ing. Raluca Tudorie
www.scr.cimr.pub.ro

Timere

Un timer este un ceas intern oferit de sistemul de operare al controllerului robot. Cu ajutorul său se pot cronometra diverse operații sau se pot genera perioade de așteptare de valoare fixă. Valoarea timerelor este dată în secunde, iar rezoluția maximă care se poate atinge este de o milisecundă.

Utilizatorul are la dispoziție 15 timere (numerotate de la 1 la 15). De asemenea, există timere cu id-uri negative, accesibile doar read-only. Acestea măsoară timpul trecut de la punerea sistemului în funcțiune, folosind diferite precizii și intervale de roll-over, descrise detaliat în V^+ Language Reference Guide.

Timerele utilizator sunt resurse globale și pot fi scrise / citite din orice program de pe orice task; de aceea, dacă mai mulți ingineri dezvoltă programe pe același controller, trebuie să se pună de acord pentru a nu-și reseta timerele între ei. O altă soluție este folosirea timerelor cu id-uri negative acolo unde este posibil, deoarece acestea sunt read-only.

Timerele pornesc odată cu sistemul, și nu pot fi oprite (deci nici repornite). Singurele operații care se pot face pe un timer sunt citirea (pentru orice timer) și scrierea (pentru timerele cu id pozitiv).

Citirea unui timer

Valoarea unui timer (în secunde) se obține folosind *funcția* `TIMER`, având ca argument id-ul timer-ului dorit:

```
TIMER(id)
```

Exemplu de utilizare:

```
IF TIMER(1) > 10 THEN
   ...
END
```

Scrierea unui timer

Atribuirea unei anumite valori (în secunde) unui timer cu id pozitiv se face folosind *instrucțiunea program* `TIMER`:

```
TIMER id = valoare
```

Exemplu:

```
TIMER 1 = 5.5
```

Cronometrarea unor operații

Pentru cronometrarea timpului necesar realizării unei serii de operații se poate folosi codul următor:

```
TIMER 1 = 0
; grup operații de cronometrat
TYPE "Timpul de executie a fost de ", TIMER (1), " secunde."
```

Generarea unei perioade de așteptare

Există mai multe metode de generare a unei perioade de așteptare, fiecare cu o anumită precizie (deci și o anumită încărcare a procesorului).

Dacă nu rulăm mai multe taskuri în paralel, putem obține precizia maximă de o milisecundă prin metoda următoare:

```
TIMER 1 = 0
WAIT TIMER(1) > delay
```

sau

```
TIMER 1 = -delay
WAIT TIMER(1) > 0
```

Metoda de mai sus utilizează un procent foarte mare din resursele procesorului doar pentru așteptare, iar acest lucru este important în special atunci când pe sistem rulează mai multe taskuri utilizator.

Putem face o temporizare cu precizie redusă, de 16 milisecunde (un ciclu sistem V^+), însă eficientă din punct de vedere al încărcării procesorului:

```
TIMER 1 = 0
WHILE TIMER(1) < delay DO
    WAIT
END
```

Sau, mai simplu, folosind instrucțiunea `WAIT.EVENT`:

```
WAIT.EVENT , delay      ; e corect cu virgulă :)
```

Exercițiu

Rescrieți exemplele de mai sus folosind timere cu id negativ.

Semnale I/O digitale

Semnalele I/O digitale se împart în două categorii: intrări și ieșiri. Aceste semnale au valori binare (`ON / OFF` sau `TRUE / FALSE` în V^+). Id-ul și tipul unui semnal ce se dorește implementat se va obține din documentația aferentă portului de I/O de pe care se dorește realizarea implementării.

Conversia la numere întregi și invers, în V^+:

```
TRUE  => -1              orice diferit de 0 => TRUE
FALSE =>  0                                 0 => FALSE
```

Din punct de vedere al conexiunii electrice, semnalele digitale respectă standardul industrial $0 \div 24$ Vcc.

Clasele de semnale suportate de V^+:

- Semnale de intrare: de la 1001 la 1012, de la 1033 la 1512.
- Semnale de ieșire: de la 1 la 8, de la 33 la 512.
- Semnale software interne (de intrare/ieșire): de la 2001 la 2512.

Semnalele interne pot fi folosite ca variabile globale boolene, pentru comunicația între task-uri.

Intrări digitale

O intrare digitală permite citirea unui semnal (informație binară) de la un echipament din exteriorul sistemului. Schema de principiu a unei intrări digitale este dată în Fig. 5.1.

Un semnal de intrare poate fi *doar citit*.

Citirea unei intrări digitale se face cu funcția `SIG`, având ca argument lista semnalului (semnalelor) citite:

```
SIG(id_semnal)
```

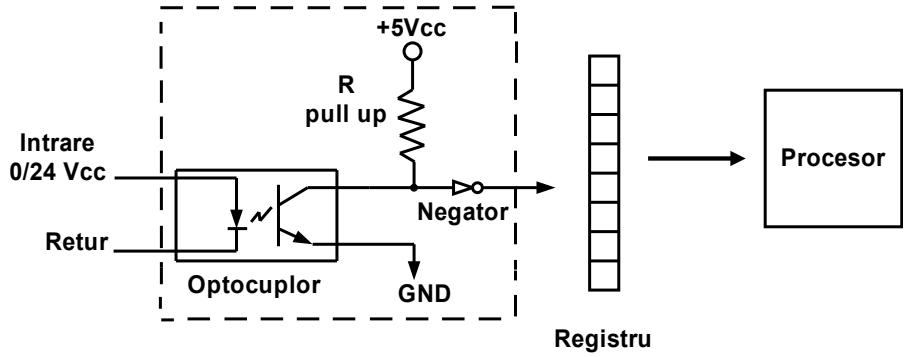

Figura 5.1: *Schema electronică pentru o intrare digitală*

Exemplu:

```
in.machine_busy = 1004
...
IF SIG(in.machine_busy) THEN
    ...
END
```

Citirea unui semnal în logică negativă se face prin schimbarea semnului id-ului:

```
SIG(-id_semnal)      echivalent cu:    NOT SIG(id.semnal)
```

`SIG` poate primi mai multe id-uri de semnale ca argumente; în acest caz, va returna operația `AND` între valorile logice ale intrărilor testate:

```
SIG (id_semnal_1, ..., id_semnal_n)
```

este echivalent cu:

```
SIG (id_semnal_1) AND ... AND SIG(id_semnal_n)
```

`SIG` acceptă ca argument(e) orice tip de semnal (intrare, ieșire, software). Se recomandă să nu se abuzeze de facilitatea de a citi un semnal de ieșire.

Ieșiri digitale

O ieșire digitală permite setarea / resetarea unui semnal binar transmis către un echipament conectat la sistem (în exterior), după schema de principiu din Fig. 5.2.

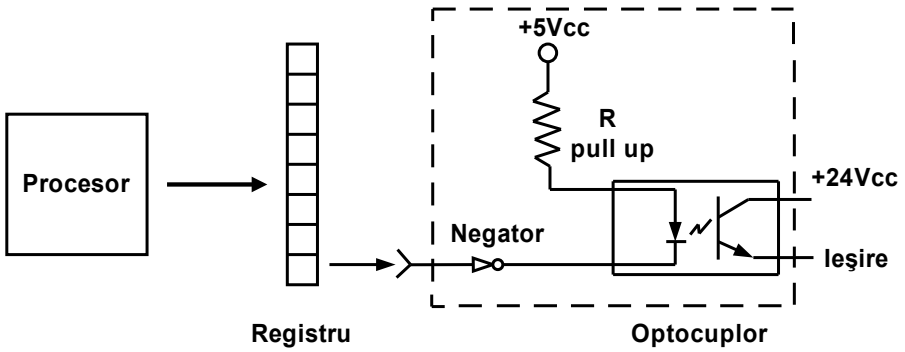

Figura 5.2: *Schema electronică pentru o ieșire digitală*

Setarea unui semnal digital de ieșire se face cu instrucțiunea program `SIGNAL`, urmat de id-ul semnalului (semnalelor) dorit(e):

```
SIGNAL id_semnal
```

Exemplu:

```
out.door_close = 5
SIGNAL out.door_close
```

Resetarea unui semnal se face prin negarea id-ului:

```
SIGNAL -id_semnal
```

Se pot seta / reseta mai multe semnale cu aceeași instrucțiune:

```
SIGNAL id_semnal_1, -id_semnal_2, id_semnal_3
```

Un semnal de ieșire poate fi (în mod normal) scris. De asemenea, un semnal de ieșire poate fi citit cu `SIG`; în acest caz se va returna ultima valoare logică scrisă de către program în acel semnal, adică valoarea memorată în buffer-ul circuitului de ieșire.

Se recomandă să nu se abuzeze de facilitatea de a citi starea unui semnal de ieșire, deoarece lizibilitatea programului poate fi afectată.

`SIGNAL` acceptă ca argument(e) doar semnale de ieșire sau semnale software (interne). Semnalele de intrare nu pot fi scrise!

Desfășurarea laboratorului

Se vor folosi robotul Cobra 600TT (postul 1) și automatul programabil al celulei de fabricație. Interfața manuală de control a celulei de fabricație cu ajutorul automatului este prezentată în Fig. 5.3:

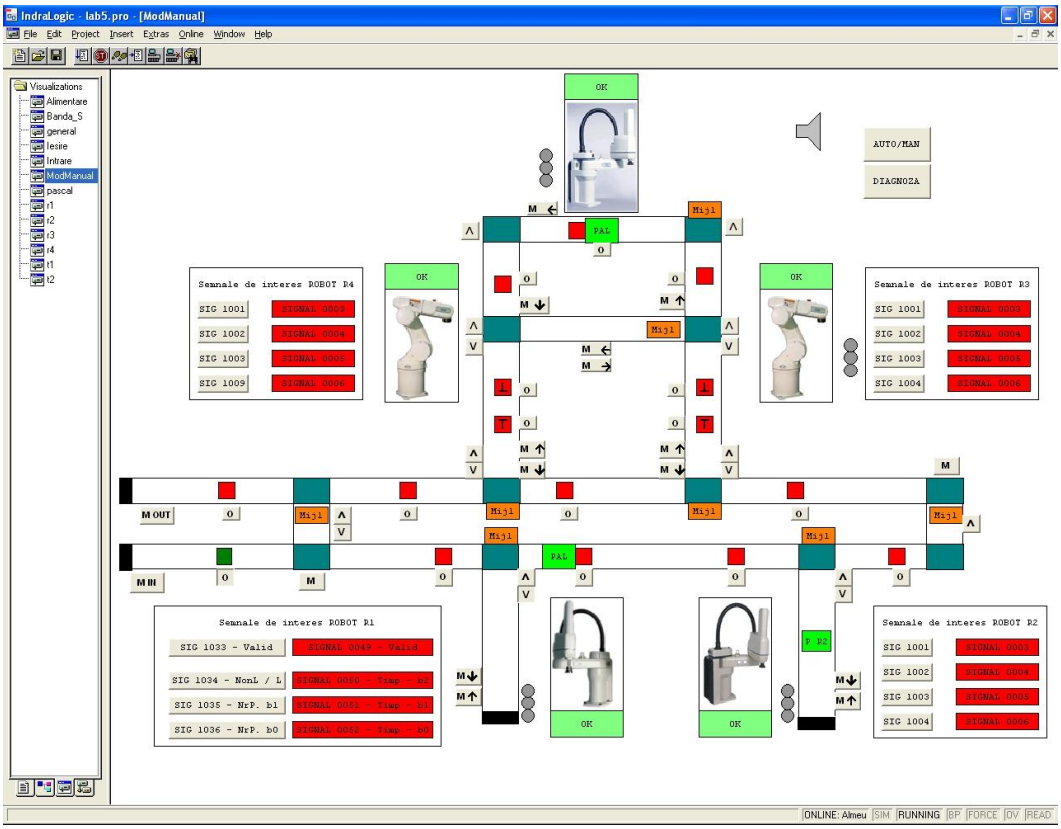

Figura 5.3: *Interfața de control automat a celulei de fabricație*

Cu ajutorul interfeței de control manual se pot comanda toate elementele celulei și se poate observa starea tuturor intrărilor automatului.

După cum se poate observa și în Fig. 5.3, între postul 1 și automat au fost implementate și afișate, din punct de vedere al robotului, 4 intrări digitale, cu ID-urile $1033 \div 1036$ (dacă butonul este apăsat atunci semnalul este setat, dacă nu este apăsat, atunci este resetat) și 4 ieșiri digitale, cu ID-urile $49 \div 52$ (dacă fondul este verde atunci semnalul este setat, dacă este roșu atunci este resetat).

Aplicația 1

Se dorește setarea / resetarea ieșirii digitale 49 la un interval de 3 secunde, apoi se dorește micșorarea intervalului inițial de două ori, de trei ori, ș.a.m.d. până se ajunge la un interval de 10 ori mai mic (0.3 secunde).

Starea semnalului se va observa pe interfața automatului programabil și pe coloana de lumini.

```
.PROGRAM delay()

   ; Laborator 5 - Test pentru semnale si timere

   AUTO i, delay

   FOR i = 1 TO 10

      delay = 3/i

      SIGNAL 49
      TIMER 1 = 0
      WAIT TIMER(1) > delay

      SIGNAL -49
      TIMER 1 = 0
      WAIT TIMER(1) > delay

   END

.END
```

Aplicația 2 - Comunicație cu automatul programabil

Se presupune că automatul realizează aducerea unei palete în Postul 1. După ce paleta a ajuns, automatul va cere robotului să plaseze pe paletă un număr de $1 \leq n \leq 3$ piese de tip "L" sau "non-L". După executarea sarcinii, robotul va transmite automatului durata care a fost necesară pentru efectuarea sarcinii, $0 \leq d \leq 7$ secunde.

Definirea semnalelor:

- 1033 - comandă începerea sarcinii pentru robot și confirmă validitatea semnalelor $1034 \div 1036$;
- 1034 - determină tipul de piese dorit; `FALSE` pentru tipul "L", `TRUE` pentru tipul "non-L";
- 1035, 1036 - numărul de piese dorit, codificat pe 2 biți (1036 = LSB);
- 49 - anunță încheierea sarcinii robotului și confirmă validitatea semnalelor $50 \div 52$;
- 50, 51, 52 - numărul de secunde necesar executării sarcinii, reprezentat ca întreg codificat pe 3 biți (52 = LSB)

```
.PROGRAM signal()

   ; Laborator 5 - Comunicatie cu automatul programabil

   GLOBAL l_storage, nl_storage, pal, #safe
   AUTO pick, place, storage
   AUTO p, r, h, d, n, nr
   h = 4.27
   nr = 3

   in.start_valid = 1033
   in.part_type = 1034
   in.nr_parts_b1 = 1035
   in.nr_parts_b0 = 1036

   out.done_valid = 49
   out.time_b2 = 50
   out.time_b1 = 51
   out.time_b0 = 52
```

```
PARAMETER HAND.TIME = 0.5
SPEED 100 ALWAYS
MOVET #safe, TRUE
RIGHTY

SIGNAL (-out.done_valid)
SIGNAL (-out.time_b2)
SIGNAL (-out.time_b1)
SIGNAL (-out.time_b0)

WHILE SIG(-in.start_valid) DO
  WAIT
END

; citesc numarul de piese dorit:

n = 0
IF SIG(in.nr_parts_b0) THEN
  n = n+1
END
IF SIG(in.nr_parts_b1) THEN
  n = n+2
END

; alternativa: deoarece SIG(...) returneaza 0 sau -1,
; n = ABS(2*SIG(in.nr_parts_b1) + SIG(in.nr_parts_b0))

; citesc tipul piesei:
IF SIG(in.part_type) THEN
   SET storage = nl_storage
ELSE
   SET storage = l_storage
END
```

```
   ; execut operatia si cronometrez:

   TIMER 1 = 0
   FOR p = 1 TO n
      r = nr - p + 1
      SET pick = SHIFT(storage BY 0, 0, h * (r-1))
      SET place = SHIFT(pal BY 0, 0, h * (p-1))
      CALL pick.place(pick, place)
   END

   MOVE #safe
   BREAK

   d = INT(TIMER(1))
   TYPE "Timpul necesar: ", d, " secunde."

   ; daca a durat mai mult de 7 secunde, trimit 7.
   IF d > 7 THEN
       d = 7
   END

   ; decodific valoarea lui d (pe 3 biti):
   IF d BAND 4 THEN        ; BAND = AND pe biti
       SIGNAL (out.time_b2)
   END

   IF d BAND 2 THEN
       SIGNAL (out.time_b1)
   END

   IF d BAND 1 THEN
       SIGNAL (out.time_b0)
   END

   ; semnalele care indica valoarea lui d sunt valide:
   SIGNAL (out.done_valid)

.END
```

Aplicația 3 - Cooperare între doi roboți

Roboții Cobra 600TT și s600 au un spațiu comun de lucru, conform Fig. 5.4. Între cei doi roboți sunt conectate semnale de I/O, având id-urile și semnificațiile din Fig. 5.4. Se dorește preluarea a 5 piese dintr-o stivă din depozitul propriu al fiecărui robot (punctul `pick_coop` învățat la baza stivei) și depunerea pieselor, în mod alternativ, într-o stivă aflată în spațiul de comun de lucru (punctul `place_coop` învățat la baza stivei), evitând coliziunea între cei doi roboți.

Folosind comanda monitor `HERE` au fost învățate punctele de precizie `#half_way` plasate la limita intersectării spațiilor de lucru și punctele `#safe` aflate în afara spațiului de lucru. Aplicațiile încep și se termină în punctele `#safe` cu gripperele deschise.

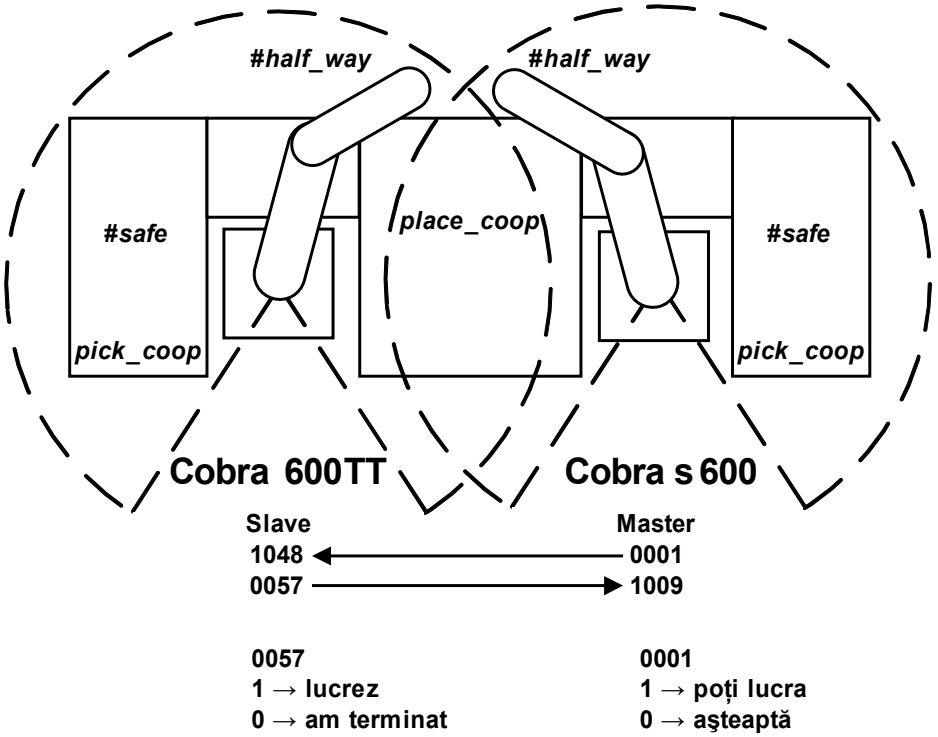

Figura 5.4: *Spațiul comun de lucru și semnalele roboților*

Robotul Cobra s600 va pune prima piesă. Inițial (înainte de pornirea programelor), semnalele de ieșire sunt `OFF`, adică spațiul comun de lucru este ocupat de către robotul Cobra S600 (`SIG(0001) = FALSE` ⇒ resursa ocupată).

Programul în V^+ pentru robotul Cobra s600 este listat în continuare. Programul pentru Cobra 600TT este similar.

```
.PROGRAM coop()

   ; Laborator 5 - Cooperare intre doi roboti

   GLOBAL pick_coop, place_coop, #half_way, #safe
   GLOBAL out.busy, in.other_busy
   AUTO pick, place

   AUTO nr_piese, p, r, h

   nr_piese = 5
   h = 4.3

   PARAMETER HAND.TIME = 0.2
   SPEED 100 ALWAYS

   out.busy = -1           ; pe logica negativa
   in.other_busy = 1009    ; pe logica pozitiva

   FOR p = 1 TO nr_piese
      r = nr_piese - p + 1
      SET pick = SHIFT(pick_coop BY 0, 0, h*(r-1))
      SET place = SHIFT(place_coop BY 0, 0, 2*h*(p-1))
      CALL pick_place_coop(pick, place)
   END

   MOVE #safe

.END
```

```
.PROGRAM pick_place_coop(pick, place)
   GLOBAL out.busy, in.other_busy

   LEFTY
   OPEN
   APPRO pick, 50
   BREAK
   SPEED 20
   MOVES pick
   CLOSEI
   DEPARTS 50
   BREAK

   MOVE #half_way
   BREAK

   ; Atentie, cursa critica!
   WAIT SIG(-in.other_busy)
   SIGNAL out.busy

   RIGHTY
   APPRO place, 50
   BREAK
   SPEED 20
   MOVES place
   OPENI
   DEPARTS 50
   BREAK
   MOVE #half_way
   BREAK

   SIGNAL -out.busy
.END
```

Pentru robotul Cobra 600TT, declarațiile semnalelor sunt:

```
   out.busy = 57           ; pe logica pozitiva
   in.other_busy = -1048   ; pe logica negativa
```

iar configurația brațului în zona comună este `LEFTY`.

Rularea programului:

- Se setează viteza 20 monitor: `.speed 20` în consolă (nu mai mult!);
- Se resetează toate semnalele de ieșire, pe ambii roboți, folosind comanda monitor `.reset`;
- Se pornește programul `coop` pe Cobra 600TT și se așteaptă până când robotul ajunge în `#half_way` și așteaptă;
- Se pornește programul `coop` pe Cobra s600.

Programul prezentat este simplificat foarte mult, deoarece are un scop pur didactic. Are însă următoarele dezavantaje:

- Dacă roboții au viteze diferite de lucru, de exemplu unul lucrează cu 50% și altul cu 10%, nu vor mai pune piesele alternativ, ci robotul mai rapid va pune mai multe piese. Robotul mai lent nu va ști asta, și va forța piesele.
- De asemenea, când vitezele de lucru sunt diferite poate apărea o situație de cursă critică, în care amândoi roboții "cred" ca au acces la spatiul comun ⇒ coliziune!
- Dacă se desface legătura fizică dintre semnalele 0067 → 1009, robotul s600 va crede că zona comună este liberă tot timpul ⇒ coliziune!
- Dacă unul din roboți este oprit forțat (Emergency Stop), semnalul lui de ieșire poate rămâne "agățat" în starea "celălalt robot poate lucra". Celălalt robot va "asculta" acest semnal și va lucra fără întrerupere. De asemenea, un semnal "agățat" înaintea începerii programelor va cauza probleme.
- Ce se întâmplă dacă apar perturbații electrice pe semnalele digitale folosite în acest program?
- Ce se întâmplă dacă operatorul nu respectă întocmai pașii necesari pentru execuția celor două programe? De exemplu, dacă uită să reseteze semnalele de ieșire? Dar dacă se pornește mai întai întâi programul pe Cobra s600?

Din aceste motive *nu* se va rula programul `coop` cu viteză ridicată!

Exercițiu

Propuneți soluții pentru a îmbunătăți robustețea aplicației de cooperare.

The end ☺