Programmation multi-tâches

cours n°5
LI326

Composant

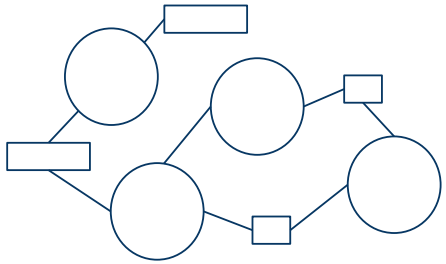
- Une interface vers les signaux
- Des paramètres de configuration
- Une mémoire d'état interne
- Des fonctions de comportement
  - initialisation
  - normal
  - exception

Signal

- Un signal est défini
  - un bit indiquant l'état du signal
    - ready / busy
    - full / empty
  - un buffer de données

Application

Nous allons imaginer les applications comme des composants matériels ou logiciels interconnectés par des signaux.



Pour :

- disposer d'une bibliothèque de composants réutilisables
- s'adapter à la nature asynchrone des comportements
- permettre une évolution simple par ajout de composants

Ordonnancement

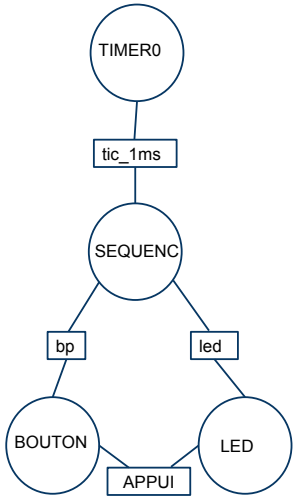
- Les composants toujours actifs.
- Le processeur exécute les composants à tour de role.
- Un composant rend le processeur quand :
  - Il tente d'utiliser un signal qui n'est pas dans l'état attendu
  - Il l'a gardé depuis trop longtemps
- C'est un ordonnancement FIFO
  - Il n'y a pas de préemption du processeur
    - parce que c'est une perte de temps
    - parce que cela permet de ne pas stocker de contexte (un composant sait quand il perd le processeur)
    - parce que c'est impossible avec le pic16

Description d'une application

```
list    p=p16f877 , n=0
include "p16f877.inc"
include "osno.inc"
include "timer0.inc"
include "sequenceur.inc"
include "bp.inc"
include "led.inc"
include "emet_char.inc"
include "lcd.inc"

CONFIGURE _CP_OFF&_WDT_OFF&_PWRTE_ON&_HS_OSC&_LVP_OFF

OBANK   BANK4
MEM     BUFF

MEM     EVENT_TMR0
BIT     EVENT_TMR0, e_tic1ms_0
BIT     EVENT_TMR0, e_tic1ms_1
BIT     EVENT_TMR0, e_tic1ms_2

MEM     EVENT_TASK
BIT     EVENT_TASK, e_bp0
BIT     EVENT_TASK, e_bp1
BIT     EVENT_TASK, e_led
BIT     EVENT_TASK, e_appui0
BIT     EVENT_TASK, e_appui1
BIT     EVENT_TASK, e_TX

DEF_TIMER0      3, EVENT_TMR0
DEF_SEQUENCEUR  PAGE0, BANK0, e_tic1ms_0, e_bp0, D'20'
DEF_SEQUENCEUR  PAGE0, BANK0, e_tic1ms_1, e_bp1, D'20'
DEF_SEQUENCEUR  PAGE0, BANK0, e_tic1ms_2, e_led, D'200'
DEF_BP          PAGE0, BANK0, e_bp0, e_appui0,  PORTB, 4
DEF_BP          PAGE0, BANK0, e_bp1, e_appui1,  PORTB, 5
DEF_LED         PAGE0, BANK0, e_led, e_appui0,  PORTB, 1
DEF_EMET_CHAR   PAGE0, BANK0, e_appui1, e_TX, BUFF
DEF_LCD         PAGE0, BANK0, e_TX, BUFF

SCHEDULE
end
```

```
DEF_LED macro   @PAGE, @BANK, @E0, @APPUI, @PORT, @PIN

    place   @PAGE, @BANK

    local   @TEST_CLIGNOTE, @STATE, @LED, @ON, @OFF, @LEDOFF

    MEM     @STATE
    BITDEF  @PORT, @PIN, @LED

    INIT
        clrf    @STATE
        banksel BANK1
        BCLR    @LED

    RUN
        BTSS    @E0             ; attend un réveil
        return
        BCLR    @E0

        BTSS    @APPUI          ; test appui
        goto    @TEST_CLIGNOTE  ; si pas d'appui
        BCLR    @APPUI          ; si appui
        comf    @STATE,f        ; changer d'état

        LABEL @TEST_CLIGNOTE
            incfsz  @STATE,W    ; saute si @STATE=0xFF
            goto    @LEDOFF       ; sort si pas de clignotement
            BNEG    @LED
            return

        LABEL @LEDOFF
            BCLR    @LED
    endm
```

Exemple



Les composants sont des automates

Un automate :

- des signaux d'entrée
- des signaux de sortie
- un état interne : STATE
- une fonction de transition qui définit la valeur de l'état interne en fonction de l'état interne et des entrées.
- une fonction de génération qui définit la valeur des sorties en fonction de l'état interne et des entrées

En logiciel

STATE = transition (STATE, entrées)
sorties = generation (STATE, entrées)

```
call   transition
movwf state
goto  generation
```

```
transition
    gotor state
        goto etat1t
        goto etat2t

generation
    gotor state
    goto etat1g
    goto etat2g
```

```
etat1t
...
    …
    retlw etatx

etat1g
    …
    …
    return
```