UE VLSI
cours 8: Machine à état (fsm) et tâches

Jean-Lou Desbarbieux
UMPC 2015

Météo

Franck est malade suite à son WE je le remplace pour le cours !

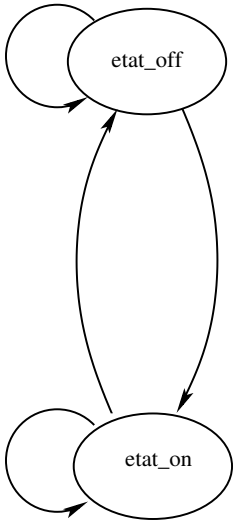
Sommaire

Objectif

Structuration du code.

- La description sous forme de MAE/FSM (Rappels).
- Support du parallélisme.
- Pas de préhemption.
- Taxinomie des transitions indispensables.

Tâche



Code C correspondant

```
void TaskLedRun()
{
Led *led= (Led *) cur_task->tdata;

if (cur_task->state == NULL ) goto etat_off;
goto *cur_task->state;

etat_off :
digitalWrite(led->pin, LOW);
if (cond) cur_state = etat_on;
return;

etat_on :
digitalWrite(led->pin, HIGH);
if (cond) cur_state = etat_off;
return;
}
```

Déclaration

Nous avons défini en C un type tâche :

```
typedef struct task_
{
char *inst_name;
unsigned long start_date;
void *state;
void (*setup) (struct task_ *t, void *arg);
void (*run) ();
void (*del) (struct task_ *t);
void *tdata;
struct task_ *next;
}Task;
```

Méthodes

Les tâches sont chaînées entre elles et une variable globale `cur_task` pointe en permanence sur la première tâche prête à être exécutée ou en cours d'éxécution.

```
extern Task *cur_task;
```

Trois méthodes régissent la vie des tâches, la philisophie objet est bien présente :

```
Task *CreateTask(char *ints_name,
    void (*setup)(Task *t, void *arg),
    void (*run)(), void (*del)(Task *t),
    void *arg);
void AddTask(Task *t);
void DeleteTask(Task *t);
```

Dans notre contexte ou la durée de vie des tâches est infinie, le destructeur a peu de chance d'etre invoqué.

Signaux

Nous avons mis en place des signaux dont la fonction est de permettre la synchronisation entre les tâches :

```
typedef struct signal_
    {
    int value;
    Task *wait_task;
    }Signal;
```

Une seule tâche peut être en attente sur un signal, l'émission et la réception sont asynchrones.

Transitions

Seulement 4 primitives pour les changements d'état :

void NextState(**void** ∗label) Transition sans condition provoque juste l'allocation du processeur à la tâche suivante.

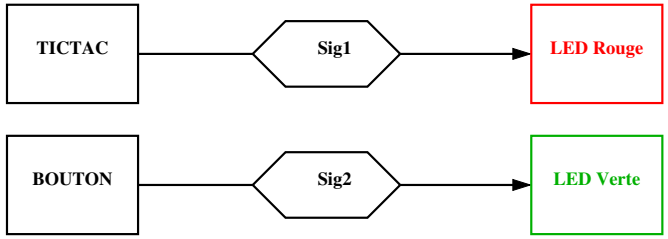
void NextStateAfter(**void** ∗label , **unsigned long** delay) Transition sans condition, mais la tâche ne reprendra son exécution qu'après un délai défini en μs.

Transitions suite

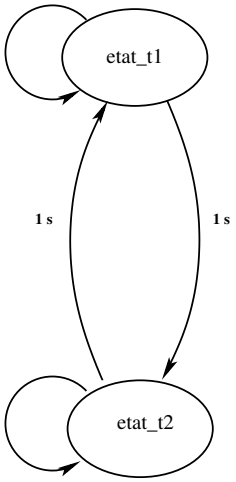
void NextStateWhenInt(**void** ∗label, **int** internum, **int** event) Transition conditionnée à la survenue d'un événement sur une des broches connectables à des une interruption. Attention ne pas se mettre en attente d'un événement qui ne surviendra jamais donc dans la machine à état tester au préalable la valeur de broche.

void NextStateWhenSig(**void** ∗label, Signal ∗s) Transition conditionnée à l'émission d'un signal par une autre tâche. L'émission et la réception des signaux sont asynchrones. Une unique tâche peut se mettre en attente sur un signal mais les émissions peuvent précéder les réceptions.

On veut réaliser un petit montage constitué d'une carte Arduino mega, un bouton poussoir et 2 leds (rouge et verte).
La led rouge clignote en permanence à la fréquence $0,5H_z$, la led verte verte s'allume exclusivement quand le bouton est enfoncé.
Le problème va être décomposé en 4 tâches : Les leds, le bouton et une tâche périodique. Nous utiliserons 2 signaux.



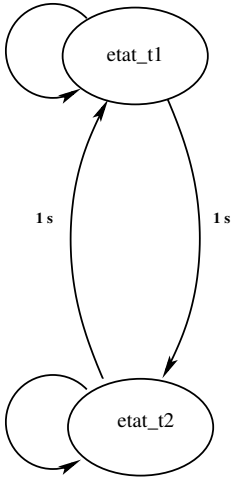
Tictac



Code C correspondant

```
typedef struct tictac_
{
unsigned long t1;
unsigned long t2;
Signal *sig;
} Tictac;

void TaskTictac(Task *t, void *arg)
{
Tictac *tictac_arg = (Tictac *) arg;
Tictac *new_tictac = (Tictac *)
                malloc(sizeof(Tictac));

*new_tictac = *tictac_arg;
t->start_date = 0;
t->tdata = new_tictac;
}
```

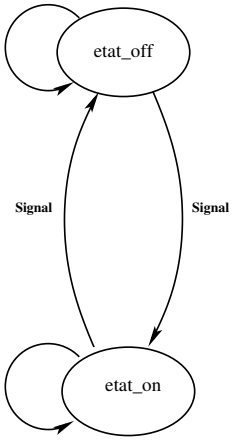
Tictac suite



Code C correspondant

```
void TaskTictacRun()
{
Tictac *tictac= (Tictac *) cur_task->tdata;

if (cur_task->state == NULL ) goto etat_t1;
goto *cur_task->state;

etat_t1 :
SendSignal(tictac->sig);
NextStateAfter(&&etat_t2, tictac->t1);
return;

etat_t2 :
SendSignal(tictac->sig);
NextStateAfter(&&etat_t1, tictac->t2);
return;
}

/* Le destructeur par defaut fera l affaire */
```

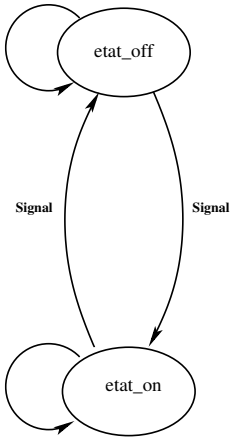
Led



Code C correspondant

```
typedef struct led_
{
int pin;
Signal *sig;
} Led;

void TaskLed(Task *t, void *arg)
{
Led *led_arg = (Led *) arg;
Led *new_led = (Led *)
malloc(sizeof(Led));

*new_led = *led_arg;
t->start_date = 0;
t->tdata = new_led;

pinMode(new_led->pin, OUTPUT);
}
```

Led suite



Code C correspondant

```
void TaskLedRun()
{
Led *led= (Led *) cur_task->tdata;

if (cur_task->state == NULL ) goto etat_off;
goto *cur_task->state;

etat_off :
digitalWrite(led->pin, LOW);
NextStateWhenSig(&&etat_on, led->sig);
return;

etat_on :
digitalWrite(led->pin, HIGH);
NextStateWhenSig(&&etat_off, led->sig);
return;
}
```

Bouton

Code C correspondant

```
typedef struct bouton_
{
int int_num;
Signal *sig;
} Bouton;

void TaskBouton(Task *t, void *arg)
{
Bouton *bouton_arg = (Bouton *) arg;
Bouton *new_bouton = (Bouton *) malloc(sizeof(Bouton));

*new_bouton = *bouton_arg;
t->start_date = 0;
t->tdata = new_bouton;

pinMode(int_pin[new_bouton->int_num], INPUT);
digitalWrite(int_pin[new_bouton->int_num], HIGH);
}
```

Bouton suite

```
void TaskBoutonRun()
{
Bouton *bouton= (Bouton *) cur_task->tdata;

if (cur_task->state == NULL ) goto etat_release;
goto *cur_task->state;

etat_release :
if (digitalRead(int_pin[bouton->int_num]) == LOW) NextState(&&etat_tempp);
else NextStateWhenInt(&&etat_tempp, bouton->int_num, FALLING);
return;

etat_tempp :
SendSignal(bouton->sig);
NextState(&&etat_press);
return;

etat_press :
if (digitalRead(int_pin[bouton->int_num]) == HIGH) NextState(&&etat_tempr);
else NextStateWhenInt(&&etat_tempr, bouton->int_num, RISING);
return;

etat_tempr :
SendSignal(bouton->sig);
NextState(&&etat_release);
return;
}
```

Setup et Loop

```
Signal Sig1={0, NULL};
Signal Sig2={0, NULL};

void setup() {
Bouton b1={2, &Sig1};
Tictac tt={1000000, 1000000, &Sig2};
Led ledr={27, &Sig2};
Led ledv={29, &Sig1};

AddTask(CreateTask((char *)"Bouton", TaskBouton, TaskBoutonRun,
                                        NULL, &b1));
AddTask(CreateTask((char *)"Tictac", TaskTictac, TaskTictacRun,
                                        NULL, &tt));
AddTask(CreateTask((char *)"Led Rouge", TaskLed, TaskLedRun,
                                        NULL, &ledr));
AddTask(CreateTask((char *)"Led Verte", TaskLed, TaskLedRun,
                                        NULL, &ledv));}
void loop()
{
TaskLoop();
}
```

Projet

Pour vendredi il faut :

- Choisir les sujets,
- définir la liste des composants.