PROG3 – Practicum Week4

Datum: 07-10-2012
Versie: 1.0
Auteur: Sander van Heumen

Embedded C - 8051
Superloop, Time-Triggered Architecture
Co-operative scheduler, UML State Machine Design

Inleiding

Lever de opdrachten van vorige week in, en stel vragen bij onduidelijkheden.

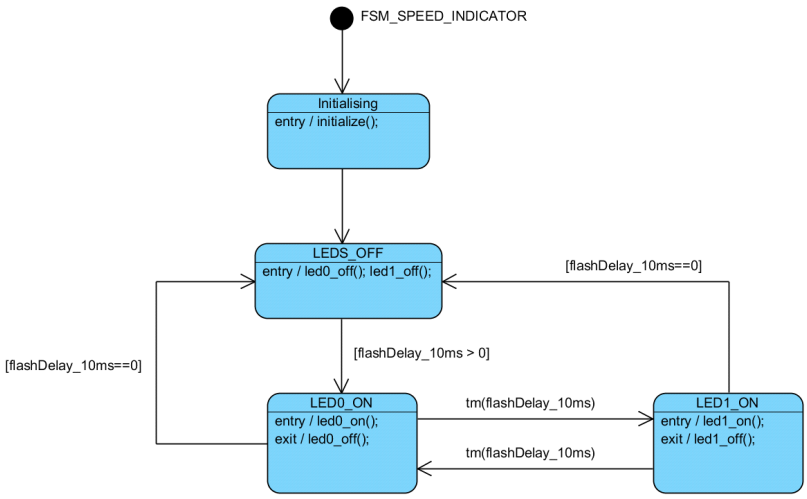
Alle opgaven van deze week maken we in **Keil:** we zijn dus "**embedded** bezig".

1. Simpele State Machine – Speed Indicator

Tijdens het hoorcollege is je hopelijk duidelijk geworden wat een State Machine is en hoe je deze kunt maken in C. In deze opgaven ga je een 'speed-indicator' maken aan de hand van 2 ledjes (ADuC847: **P0^0==LED0** en **P0^1==LED1**). De speed-indicator geeft de snelheid aan van een denkbeeldige elektromotor. Wanneer de motor op volle snelheid draait, moeten de LEDS om de 10ms worden getoggled. Wanneer de motor niet draait, moeten beide LEDS uiteraard uitstaan. De snelheid van onze denkbeeldige motor heeft 5 standen, welke door een globale variabelen (in het StateDiagram hieronder afgebeeld noemen we deze `flashDelay_10ms`) moet kunnen worden ingesteld:

Toggle snelheid LEDS	Procentuele snelheid motor
LED0 en LED1 toggelen met een snelheid van 10ms	Motor draait op volle toeren (100%).
LED0 en LED1 toggelen met een snelheid van 50ms	Motor draait op 80%.
LED0 en LED1 toggelen met een snelheid van 250ms	Motor draait op 60%.
LED0 en LED1 toggelen met een snelheid van 500ms	Motor draait op 40%.
LED0 en LED1 toggelen met een snelheid van 1seconden	Motor draait op 20%.
LED0 en LED1 staan beide uit.	Motor staat stil.

Het **UML State Diagram** van deze snelheids-indicator is hieronder weergegeven:



Figuur 1 - UML State Diagram Diagram of 'speed-indicator'

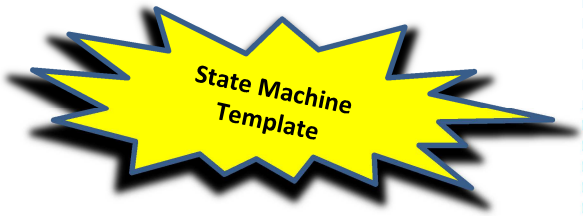
De statemachine die je dient te maken creëer je altijd in **2 aparte bestanden:**

`my_statemachine.h`	• Hierin staat het functieprototype van de statemachine. Dit is niets meer en niets minder dan een *`void-void`* functie waarin de statemachine is opgebouwd (in de functiedefinitie zit dus de `switch` en `case` structuur). • Het functieprototype van een **optionele** initialisatie routine van de statemachine. • De functieprototypes van de functiedefinities welke in `my_statemachine.c` staan EN welke je openbaar toegankelijk wilt maken. • Alle typedefinities die je gebruikt in `my_statemachine.c` EN welke je openbaar toegankelijk wilt maken.
`my_statemachine.c`	• Alle functiedefinities die je nodig hebt om de statemachine te bouwen. • De typedefinities van je statevariabelen. Deze statevariabelen worden alleen gebruikt in de statemachine beschreven in dit bestand, dus we maken het type niet globaal, maar houden het fijn lokaal (daarom staat het **niet** in de header file). • De functie-prototypes van de functies die we **alleen** in deze statemachine mogen/willen/kunnen gebruiken. “Uiteraard staat dit prototype boven de eerste functieaanroep in dit bestand”. Ook hier geldt weer, de functies horen alleen bij *deze* statemachine, en horen nergens anders bij, houdt daarom de deuren gesloten voor andere (kwaadwillige…) gebruikers!

```
#ifndef MY_STATEMACHINE_H
#define MY_STATEMACHINE_H

#include "main.h"
#include "port.h"

/* place necessary include files below here */

/* declare (PUBLIC) function prototypes below here */
void statemachine (void);
void init_statemachine (void);

#endif
```

Figuur 2 - `my_statemachine.h`

Hieronder is de inhoud weergegeven van `my_statemachine.c`, waarin de **template** van de UML State Machine zit verstopt:




```c
#include "my_statemachine.h"

/* define your FSM states below here */
enum eSystem_State{STATE1, STATE2, STATE3};
typedef enum eSystem_State eSystem_State;

/* define your (PRIVATE) function prototypes below here */

/* FSM entry */
void statemachine (void)
{
    /* statemachine control */
    static eSystem_State currentState = STATE1; // state-variable
    static bit stateEntry = TRUE;               // true upon entering a new state
    eSystem_State nextState  = currentState;

    /* declare local user-variables below here */


    /* statemachine */
    switch (currentState) {
        case STATE1:
            if (stateEntry) {
                // place ENTRY actions of STATE1 here

            }
            else {
                // STATE actions; change state if needed

            }
            if (currentState != nextState) {
                // place EXIT actions of STATE1 here

            }
            break;

        case STATE2:
            if (stateEntry) {
                // place ENTRY actions of STATE2 here

            }
            else {
                // STATE actions; change state if needed

            }
            if (currentState != nextState) {
                // place EXIT actions of STATE2 here

            }
            break;

        /* create all other states below here */

        default:
            // internal error: not possible
            currentState = STATE1;
            break;
    }

    /* update FSM internals */
    if (currentState != nextState) {
        /* change of state */
        stateEntry = TRUE;
        currentState = nextState;
    }
    else if (stateEntry) {
        /* state entered */
        stateEntry = FALSE;
    }
}

/* FSM initialisation */
void init_statemachine (void)
{

}

/* define your function defintions below here */
//…
// a lot more useful code
//…
```

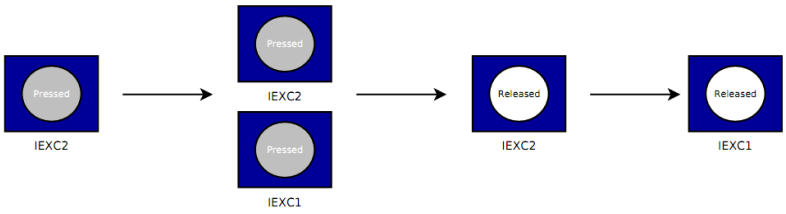
Figuur 3 - `my_statemachine.c`

Maak nu de statemachine van de speed-indicator (pagina1). Maak gebruikt van het voorbeeld project op Blackboard (../programmeren3/week4/) genaamd “**statemachine_template.zip**”.

2. Advanced State Machine – Motor-Management

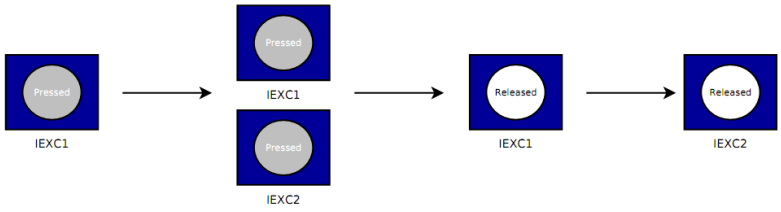
Je moet nu een motormanagement systeem bouwen. Dit inputs (*events*) van het motormanagement systeem zijn enkel 2 druktoetsen, welke IEXC1 en IEXC2 worden genoemd. Diverse combinaties van deze druktoetsen zorgen ervoor dat de motor linksom, rechtsom, sneller, langzamer of meteen tot stil stand komt. Hieronder zijn deze druk-toets-combinaties weergeven (zoom goed in, dan zie je of de toets moet worden ingedrukt resp. moet worden losgelaten).

Mode: Left (motor beweegt linksom):



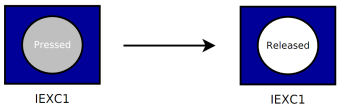
Figuur 4 - Left Mode

Mode: Right (motor beweegt rechtsom):



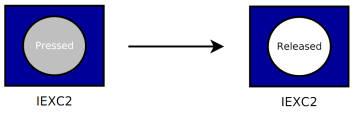
Figuur 5 - Right Mode

Mode: Speed-Up (motor versneld met 20%):



Figuur 6 - Speed-Up

Mode: Speed-Down (motor vertraagd met 20%):



Figuur 7 - Speed-Down

Mode: Brake (motor remt en komt **direct** tot stilstand):

Figuur 8 - Brake

Alle andere combinaties zijn **ongeldig** en dienen door de ontworpen statemachine te worden opgevangen.

Een druktoets wanneer ingedrukt (en losgelaten), is na 20ms stabiel. Hou hier dus rekening mee! (het rekenen hiermee houden noemen we *ontdenderen*).

De outputs (*actions*) van het motormanagement systeem zijn LEDS welke de acties van het motormanagement systeem *simuleren* (normaal gesproken koppel je aan de acties van de statemachine I/O-pinnen die bijvoorbeeld relais schakelen). Hieronder is een tabel weergegeven met de verschillende acties en de daarbij, ter indicatie, horende LEDS:

LED (on)	Overeenkomende motormanagement acties
P0^7	Motor beweegt linksom.
P0^6	Motor beweegt rechtsom.
P0^7 AND P0^6	Motor wordt direct stilgezet: “brake”
P^0 EXOR P^1	Motorsnelheid. Deze LEDS dienen te volgens de tabel op pagina 1 te worden getoggled.
P^3	Indien er een ongeldige invoer is gedecteerd, dient dit LEDje te gaan branden. Wanneer alle inputs weer zijn vrijgegeven, moet het LEDje weer uitgaan.

Je **moet** voor het maken van deze opdracht weer gebruik maken van het basisproject *in* **statemachine_template.zip**

Het versnellen resp. vertragen van de motor:

Het versnellen en vertragen (een bepaalde toets combinaties) van de motor is weergegeven in Figuur 6 - Speed-Up en Figuur 7 - Speed-Down. Wanneer de motor stilstaat en er wordt versneld, moet de motor op 20% van het maximale toerental gaan draaien. Indien er nog een keer wordt versneld op 40%. etc. Indien er wordt vertraagd moet het toerental van de motor afnemen met 20%.

Het inlezen van de schakelaars:

De code die hoort bij het detecteren van een drukactie van druktoets IEXC1 resp. IEXC2 is:

```
if (SW1_PRESSED()) {
    //switch 1 pressed
}

if (SW2_PRESSED()) {
    //switch 1 pressed
}

// De verwijzingen naar de schakelaars staan in “port.h”
```

*2.1 **Ontwerp** nu een **UML StateDiagram** welke bovenstaand motormanagement implementeert.*

*2.2 **Implementeer** nu het UML StateDiagram van je ontwerp gemaakt in vraag 2.1.*