UNIVERSITAT POLITÈCNICA DE VALÈNCIA
Antonio Martí Campoy
amarti@disca.upv.es

Library architecture

Methods and Tools for Software Quality

ESIGELEC

Introduction.

This document called "Library architecture" presents full detail for the use of the functions and constants included in the software library *robot_library.a*. The student can include this library in its project and use all the functions and constants defined in it. The library is organised in eigth modules: System; Sides.h; Display; Control_motors; Control_encoders; scheduler; Compass; and Time-out.
Each module is composed of two files, one name_of_module.c (source code is not available to students, at the moment, but functions may be called) and another file name_of_module.h that must be included in the student's source code using directive *#include "name_of_module.h".* As exception, module *Sides* offers just constants but not functions or executable code.
Following the description of each module.

Module System

The purpose of this module is to configure the system clock and derived clocks for the proper operation of CPU and internal peripherals.
Constants:
There are no constants defined.

Functions:
void Clock_graceInit_DCO_12M(void);
This function switch on the internal DCO of the microcontroller and configure it to run at 12MHz. Activate MCLK, SMCLK, and ACLK all of them at 12MHz
No parameters. No return value.

Module Sides

The purpose of this module is to define constants to name and select the sides of the robot.

Constants:
#define LEFT 0
#define RIGHT 1
#define MAX_SIDES 2 // Number of sides of the robot. Used to validate parameters

Functions:
There are no functions available.

Module Display

The purpose of this module is to allow accessing to the 4-digit, 7-segments display from any function of the system. The display is connected by means of SPI using UCA0.
The display is able to show 4 digits. The first digit is shown in the right side. When a new digit is sent, previous digit/s is/are displaced to left.

Constants:
#define CLEAR_DISPLAY 0x76 //Hex code to clear the display

Functions:

void init_display (void);

Configure UCA0 to allow sending data to the display.

No parameters. No return value.

void show_byte (uint8_t byte);

Send a numerical value as data (one digit) or command (it depends of the value) to be displayed or to control the display.

One parameter: 8-bits unsigned.

No return value.

Example of use: *show_byte (CLEAR_DISPLAY); show_byte (0);*

void show_number (uint16_t number);

Send a numerical value to be displayed.

Parameter: numerical value, 16-bits unsigned, in range 0 to 9999. Uses an internal function to fill with zeros at most significant digits.

No return value.

Example of use: *show_number (56); show_number (5678);*

*void show_string (const uint8_t *msg);*

Send a string (only four characters may be displayed) and not all characters are available.

Parameter: a const string.

No return value.

Example of use: *void show_string ("Hola");*

Module Control_motors

This module offers constants and functions to control the two motors included in the robot. Each motor is controlled independently, using PWM generated by Timer A0 and interrupts. Digital i/o ports are used to control the direction of the revolution.

Constants:

#define PWM_FREQ 1200 //Freq PWM = 10 KhZ with SMCLK = DOSC/1 = 12Mhz

Functions:

void Init_motors(void);

Configure TimerA0, interrupts and digital I/O required for motor operation. Leave motors stopped.

No parameters. No return value.

uint8_t Speed_motor (int8_t percentage, uint8_t side);

This function set the speed of one motor, in forward of reverse mode. The function is **NOT blocking**, that is, it finish as soon the motor is configured and leave the wheel running. Calling function is in charge to stop the motor or change speed as required.

Two parameters:

Uint8_t percentage: speed desired, in percentage of the max speed. Range: [-100 , 100].
If magnitude of *percentage* is below 10% then the function considers it as 0%. If magnitude *percentage* is over 90%, then is considered as a magnitude of 100%.
uint8_t side: selects the motor to run (left or right). It is strongly suggested to use the definitions of side.h
Return value:
If any of the parameters are out of range then function returns 1, else returns 0.
Example of use: *error = Speed_motor (45, LEFT); //Don't forget to check the value of error.*

void Emergency_stop (void);
It stops without delay the movement of two wheels. In order to restart the motors is mandatory to call again to *Init_motors();*
No parameters. No return value.

Module Control_encoders

This module offers constants, functions and a structure as data type to configure encoders, store distance of each wheel separately and to access the distances stored. Encoders are activated by interrupts. After configuring the encoders, they are always working.
This module only measures distance, not speed.

Constants:
#define MAX_STEPS 12 //12 activations means one complete revolution of the wheel.

Structure:
```
typedef struct struct_distance{
  uint8_t steps;
  uint16_t turns;
} distance_type;
```

Functions:
void Init_encoders_distance();
Configure I/O ports and interrupts to detect changes in encoder. Set to zero the values of steps and turns for the two wheels. Values of *steps* and *turns* are automatically increased by ISRs (interrupt service routines). When *Steps* reach value MAX_STEPS it reset to 0 and increase *turns*. No matter the direction of revolution of the wheel, *steps* and *turns* are increased. Application must deal with this issue.
No parameters. No return value.

void Clear_Distance(uint8_t side);
Set to 0 the counters (both *steps* and *turns*) for one of the wheel.
Parameters:
uint8_t side: selects the wheel to clear (left, or right). It is strongly suggested to use the definitions in side.h
No return value.

*uint8_t Read_distance (uint8_t side, distance_type *dist);*
This function allows reading the values of steps and turns of a wheel.
Parameters:
uint8_t side: selects the wheel to read (left, or right). It is strongly suggested to use the definition of side.h
*distance_type *dist:* parameter passed by reference. Pointer to a variable of type *distance_type* declared in the scope (or higher) of the calling function.
Return value:
If parameter *side* is out of range returns 1, else returns 0.
Example of use:
distance_type my_distance_left; //Variable declaration
error = Read_distance (LEFT, &my_distance_left);

Module Compass

This module offers constants and functions to read the bearing from the digital compass. Compass is accessed by means of I2C using UCB0.
Functions to read compass bearing uses LPM0 (low power mode 0: CPU off) and interrupts to exchange data with the compass.

Constants:
#define MAX_low_BEARING 255 //Maximum bearing in low (8 bits) resolution
#define MAX_BEARING 3599 //Maximum bearing in high (16 bits) resolution

Functions:
uint8_t compass_init(void);
Configure I2C to allow communicating with the digital compass.
No parameters
Return value: If it is not possible to initialize the compass, returns 1, else returns 0.

uint8_t Read_compass_8 (void);
Reads the bearing in low resolution as unsigned byte. Blocking function
No parameters.
Return value: the bearing in the range 0 to 255 for a full circle.

uint8_t Read_compass_16 (uint16_t);
Reads the bearing in high resolution as unsigned 16-bits.
It is a blocking function, but includes a time-out mechanism to exit if the compass doesn't connect. The time out is implemented using the functions described afterward in module Time-out.
Parameters: pointer to a uint16_t where the function returns the bearing in the range 0 to 3599 for a full circle.
Return value: 0 if there was no problem reading the bearing.

Module scheduler

This module offers a function called *void callback_2ms(void)*. This function is called every 2 milliseconds from the interrupt service routine (ISR) of timer TA0. This function sets to one a variable called Tick_out. This module is needed to create non-blocking movements.
Very important:
- Never call this function.
- This functions indeed as part of the ISR. In MSP430 architecture this means all interrupts disabled.
- Adding code to this function may degrade system performance.
- The initial place of variable Tick_out declaration might be the final one.

Module Time-out

This module offers a basic Time-Out mechanism based in the WatchDog Timer (WDT) working in interval mode.
This module is used by function *uint8_t Read_compass_16 (uint16_t);* so you can't use this module while reading from compass.

If the timer is running and reaches the predetermined elapse time, raises an interrupt.
The interrupt sets to one a global variable, the timer continues running **and Low Power mode is deactivated on exit.**

Constants:
There are no constants defined.

Public variables:
extern uint8_t TIME_OUT;
Initialised at 0, set to 1 when timer over.

Functions:

void init_timeout (void);

Configure WDT in Interval mode using ACLK. The interval gap is 2.7 ms.
Interrupt is enabled but the timer remains stopped.

void start_timeout(void);

Resets the global variable TIME-OUT, clear the WDT counter and runs the timer.

void stop_timeout(void);

Stops the timer and resets the global variable TIME-OUT.