ELC018 Real-time Software Engineering

Driving the washing machine

In this laboratory you will download code to the STM Discovery board (as in the Device Driver laboratory) in order to read inputs from the washing machine and to drive outputs. Basic device drivers written in C are provided and you will need to start considering how these will be translated into C++. In task 2, you will need to write all the device driver code in C++.

The washing machine interface

The washing machine is connected to a number of ports on the Discovery board. The washing machine output devices connected on port C are shown below.

Port C (`0x48000800`)	bit 6
Washing machine device	buzzer

The washing machine output devices connected on port D are shown below.

Port D (`0x48000C00`)	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 8
Washing machine device	motor direction	switch reset	7 seg D	motor control	7 seg B	7 seg C	7 seg A

The washing machine input devices connected on port E are shown below.

Port E (`0x48001000`)	bit 15	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
Washing machine device	motor speed	cancel	accept	prog 2	prog 3	door	prog 1

As in the device driver laboratory, other registers of the ports are used to configure operation. Ports C, D and E have been correctly set up in the code provided.

Checking the set up

On the Discovery target board, make sure to check the following before you start.

- The USB cable is connected to the **ST-LINK/V2 for debugging** connector on the Discovery board and to a free USB socket on the PC.
- The jumpers should have JP4 on and CN4 on.
- The ribbon cable to the washing machine is connected.

On the washing machine, make sure to check the following before you start.

- The power supply is connected, plugged in to the mains and switched on.
- The ribbon cable from the interface board is connected.
- The motor control switch near the ribbon cable socket is set to ‘port’. This is its position for normal operation.
- The speed sensor switch to the left of the ‘accept’ button is set to ‘port’.
- The DIP switches immediately to the left of the seven segment display should be in their ‘fault off’ position.
- There is a switch near the buzzer to turn it off. You may wish to switch the buzzer to its off position to prevent irritation to you and colleagues in the lab.

Entering the code

The washing machine code for the laboratory can be found on the Learn server. Although it is a C++ file, the manner of implementation of the device drivers is that normally used in C rather than C++. It will be your task to convert the C device drivers to C++ device drivers.

The code accesses each of the washing machine peripherals in turn and then simply checks whether the door is open by monitoring the door sensor. If the door is open, the motor does not run. If it is closed, the motor runs.

```
int main(void)
{
  // STM32F3 Discovery Board initialization
  board_startup();

  // configure port C
  *GPIO_C_Mode_Addr =  (uint32_t) GPIO_C_Mode;
  *GPIO_C_Speed_Addr = (uint32_t) GPIO_C_Speed;
  *GPIO_C_Pull_Addr =  (uint32_t) GPIO_C_Pull;

  // configure port D
  *GPIO_D_Mode_Addr =  (uint32_t) GPIO_D_Mode;
  *GPIO_D_Speed_Addr = (uint32_t) GPIO_D_Speed;
  *GPIO_D_Pull_Addr =  (uint32_t) GPIO_D_Pull;

  // configure port E
  *GPIO_E_Mode_Addr =  (uint32_t) GPIO_E_Mode;
  *GPIO_E_Pull_Addr =  (uint32_t) GPIO_E_Pull;

  // hold a bit value returned from a port
  bool port;

  // try out each washing machine device in turn
  *GPIO_C_Odr_Addr ^= (uint16_t) 0x0040;   // toggle PC6 buzzer
  HAL_Delay(100); // 100ms delay
  *GPIO_D_Odr_Addr |= (uint16_t) 0x1000;   // PD12 motor control - on
  HAL_Delay(100); // 100ms delay
  *GPIO_D_Odr_Addr &= ~(uint16_t) 0x8000;  // PD15 motor direction - clockwise
  HAL_Delay(1900); // 100ms delay
  *GPIO_D_Odr_Addr |= (uint16_t) 0x8000;   // PD15 motor direction - anticlockwise
  HAL_Delay(1000); // 100ms delay
  *GPIO_D_Odr_Addr &= ~(uint16_t) 0x1000;  // PD12 motor control - off
  HAL_Delay(100); // 100ms delay

  *GPIO_D_Odr_Addr &= ~(uint16_t) 0x2D00;  // turn off all 7 segments of display (display 0)
  HAL_Delay(100); // 100ms delay
  *GPIO_D_Odr_Addr |= (uint16_t) 0x2000;   // turn on bit D MSB
  HAL_Delay(100); // 100ms delay
  *GPIO_D_Odr_Addr |= (uint16_t) 0x0400;   // turn on bit C
  HAL_Delay(100); // 100ms delay
  *GPIO_D_Odr_Addr |= (uint16_t) 0x0800;   // turn on bit B
  HAL_Delay(100); // 100ms delay
  *GPIO_D_Odr_Addr |= (uint16_t) 0x0100;   // turn on bit A LSB (all outputs 1 - display blank)
  HAL_Delay(100); // 100ms delay
  *GPIO_D_Odr_Addr &= ~(uint16_t) 0x2D00;  // turn off all 7 segments of display  (display 0)
  HAL_Delay(100); // 100ms delay

  *GPIO_D_Odr_Addr |= (uint16_t) 0x4000;  // PD14 HIGH accept switch input
  HAL_Delay(100); // 100ms delay
  port = (*GPIO_E_Idr_Addr) & 0x0100 ;  // PE8  programme select 1 (rightmost)
  if (port) *GPIO_D_Odr_Addr |= (uint16_t) 0x0100; else *GPIO_D_Odr_Addr &= ~(uint16_t) 0x2D00;
  HAL_Delay(100); // 100ms delay
  port = (*GPIO_E_Idr_Addr) & 0x0200 ;  // PE9  programme select 2 (middle)
  if (port) *GPIO_D_Odr_Addr |= (uint16_t) 0x0100; else *GPIO_D_Odr_Addr &= ~(uint16_t) 0x2D00;
  HAL_Delay(100); // 100ms delay
  port = (*GPIO_E_Idr_Addr) & 0x0400 ;  // PE10 programme select 3 (leftmost)
  if (port) *GPIO_D_Odr_Addr |= (uint16_t) 0x0100; else *GPIO_D_Odr_Addr &= ~(uint16_t) 0x2D00;
  HAL_Delay(100); // 100ms delay
  port = (*GPIO_E_Idr_Addr) & 0x0800 ;  // PE11 door open/close
  if (port) *GPIO_D_Odr_Addr |= (uint16_t) 0x0100; else *GPIO_D_Odr_Addr &= ~(uint16_t) 0x2D00;
  HAL_Delay(100); // 100ms delay
  port = (*GPIO_E_Idr_Addr) & 0x1000 ;  // PE12 accept switch
  if (port) *GPIO_D_Odr_Addr |= (uint16_t) 0x0100; else *GPIO_D_Odr_Addr &= ~(uint16_t) 0x2D00;
  HAL_Delay(100); // 100ms delay
  port = (*GPIO_E_Idr_Addr) & 0x2000 ;  // PE13 cancel switch
  if (port) *GPIO_D_Odr_Addr |= (uint16_t) 0x0100; else *GPIO_D_Odr_Addr &= ~(uint16_t) 0x2D00;
  HAL_Delay(100); // 100ms delay
```

```
  port = (*GPIO_E_Idr_Addr) & 0x8000 ;  // PE15 motor speed feedback
  HAL_Delay(100); // 100ms delay
  *GPIO_D_Odr_Addr &= ~(uint16_t) 0x4000;  // PD14 LOW reset switches
  HAL_Delay(100); // 100ms delay

  // Only run the motor if the door is closed
  *GPIO_D_Odr_Addr &= ~(uint16_t) 0x8000;  // PD15 motor direction - set to clockwise
  while(1) {
    port = (*GPIO_E_Idr_Addr) & 0x0800 ;  // PE11 check if door open or closed
    if (port) {
      *GPIO_D_Odr_Addr |= (uint16_t) 0x1000;  // PD12 motor control - off
    }
    else {
      *GPIO_D_Odr_Addr &= ~(uint16_t) 0x1000;  // PD12 motor control - off
    }
  }
}
```

Running the code on the target

Build the code and run it under the debugger as in the **Device Drivers** laboratory. Satisfy yourself that the code operates as described for each of the washing machine peripherals and then check that the movement of the motor can be controlled by operating the door switch.

Modifying the example

The example has been written using C device drivers. As a starting point to the development of the C++ code, consider defining classes for the `Door` and `Motor` devices and consider what members and member functions these classes should have. These objects will actually reside in the 'device layer' as considered in the lecture on device drivers, see Fig. 1. The classes modelling the devices on the STM Discovery target board (mainly the `Ports` on the board) will be in the 'target hardware layer'. In task 2, you will need to write code for both the device layer and the target hardware layer and the application will contain your test code. In task 3, you will write the washing machine code for the application layer.

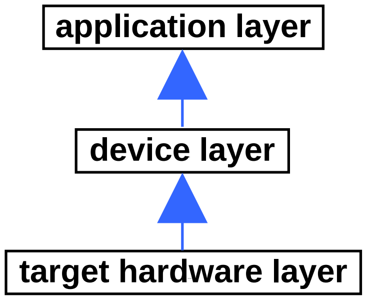

Fig 1. The three layers in the structure of embedded software