Politecnico di Torino

Operating systems for Embedded Systems

Professor Violante Massimo

Assignment #1

Student:
Luca Di Grazia
S243844

Summary

1 Introduction

This assignment consists of developing an embedded system to manage some data coming from sensors of a Weather Station. It is equipped with an anemometer, a rain sensor and a thermometer.I used some hardware devices and software tools:

- FRDM K64F
- Micrium uC/OS-III
- Kinetis Design Studio 3 IDE
- Digilent Analog Discovery 2
- Digilent waveform tool
- Tera Term

2 Configuration

2.1 Hardware configuration

I used a PTC3 pin with the Flex timer for the anemometer, PTB2 and PTB10 with ADC for analog input, accelerometer with I2C for earthquakes detector.

Table 1: Hardware configuration

Function	**PIN**	**Input**	**Hardware module**
Anemometer	PTC3	Digital	Flex timer
Rain sensor	PTB2	Analog	ADC0
Thermometer	PTB10	Analog	ADC1
Earthquakes detector	-	Board position	Accelerometer

2.2 Software architecture

2.2.1 Tasks

I used 6 tasks to perform the assignment with some functions for setting up hardware modules and for managing interrupt handlers.

Table 2: Task organization

#	**Task**	**Purpose**	**Sample time**
1	Accelerometer	reading accelerations in [g] over X/Y/Z axes	1 s
2	Thermometer	waiting for a new data from ADC1 and computing the temperature	60 s
3	Rain_sensor	waiting for a new data from ADC0 and computing the rain level	60 s
4	TaskFtm0	waiting for a new data from Flex timer and computing the wind speed	1 s
5	Zero_Hz	managing the 0 Hz wind situation	2 s
6	Serial_print_values	printing all the values computed by other tasks	1 s

The task Serial_print_values() is the only function that prints values, it transmits data every second to a serial port configured as 1152008N1 (Baud rate 115200).

```
<21> <0.75 [m/s]> <10 [mm]> <-20.0 [C]> <X= 0.02 [g]> <Y=-0.04 [g]> <Z=1.04 [g]>
<22> <0.00 [m/s]> <10 [mm]> <-20.0 [C]> <X= 0.02 [g]> <Y=-0.04 [g]> <Z=1.04 [g]>
<23> <0.00 [m/s]> <10 [mm]> <-20.0 [C]> <X= 0.02 [g]> <Y=-0.04 [g]> <Z=1.04 [g]>
<24> <0.00 [m/s]> <10 [mm]> <-20.0 [C]> <X= 0.02 [g]> <Y=-0.04 [g]> <Z=1.04 [g]>
<25> <0.00 [m/s]> <10 [mm]> <-20.0 [C]> <X= 0.02 [g]> <Y=-0.04 [g]> <Z=1.04 [g]>
<26> <0.75 [m/s]> <10 [mm]> <-20.0 [C]> <X= 0.02 [g]> <Y=-0.04 [g]> <Z=1.04 [g]>
<27> <0.75 [m/s]> <10 [mm]> <-20.0 [C]> <X= 0.02 [g]> <Y=-0.04 [g]> <Z=1.04 [g]>
<28> <0.75 [m/s]> <10 [mm]> <-20.0 [C]> <X= 0.03 [g]> <Y=-0.04 [g]> <Z=1.04 [g]>
<29> <7.50 [m/s]> <10 [mm]> <-20.0 [C]> <X= 0.03 [g]> <Y=-0.04 [g]> <Z=1.04 [g]>
```

Figure 1: Output example of BSP_Ser_Printf() in Serial_print_values Task

2.2.2 Semaphores

I used a lot of semaphores in this project; managing a semaphore causes a lot of machine cycles and a possible alternative is to enter in critical sections disabling interrupts:

```
1  CPU_CRITICAL_ENTER();
2  OSIntEnter();
3  //Critical section
4  CPU_CRITICAL_EXIT();
5  OSIntExit();
6
```

Even though semaphores require more computing time, I decided to use them because this project is an application that works in seconds so it is acceptable, another reason why I used semaphores is because I want to improve my skills building a robust solution.

Table 3: Semaphores list

#	**Semaphore**	**Purpose**
1	windfvalue_sem	protecting the global variable of wind speed
2	rainfvalue_sem	protecting the global variable of rain level
3	tempfvalue_sem	protecting the global variable of temperature
4	ftm0sem	is posted when Flex timer captures a waveform period
5	flag_sem	is used to manage the 0 Hz wind situation
6	adc0sem	is posted when ADC0 computes a conversion
7	adc1sem	is posted when ADC1 computes a conversion
8	X_sem	protecting the global variable of axis X
9	Y_sem	protecting the global variable of axis Y
10	Z_sem	protecting the global variable of axis Z

3 Wind Anemometer

The Wind Anemometer operates in the range $0\,\mathrm{m\,s^{-1}}$ to $60\,\mathrm{m\,s^{-1}}$ ($0\,\mathrm{Hz}\rightarrow 80\,\mathrm{Hz}$). It produces as output a digital square waveform with output frequency linearly proportional

to the wind speed and it is sampled every second.

3.1 Implementation in software

The first way to implement the anemometer is using some time measurement functions defined in Micrum OS. The idea is simple: an interrupt service routine is responsible for posting a semaphore any time a rising edge is detected and there is a task that waits for this semaphore twice, for the first rising edge and the second one. The difference of these two timestamps gives the frequency of the waveform. This solution is not complex to implement but it is very sensitive to the execution time of the different instructions, so i decided to implement this requirement using a dedicate hardware module.

3.2 Implementation in hardware

I used the pin PTC3 as input and flex timer FTM0 to capure the digital waveform and compute its frequency. It is not easy to work using flex timer with low frequencies, because when the timer starts counting there are some overflows, that must be saved to perform a right measurement. So for this purpose in the interrupt handler I used a global variable (set to zero for every measurement) and then I used it to compute the final result:

```
1    //C2V -> Rising edges
2    //C3V -> Falling edges
3    if( FTM0_C2V > FTM0_C3V)
4     ftm0_pulse = 0xFFFF+FTM0_C3V-FTM0_C2V + (overflow - 1)*65536;
5    else
6     ftm0_pulse = FTM0_C3V-FTM0_C2V + (overflow)*65536;
7
```

The problem was that in the 0 Hz situation the wind speed always remained unchanged. To fix this problem I used a flag that is set to 1 when pulse has been measured, a specific task(Zero_Hz()) that every 2 seconds set the flag to 0 and if flag is not restored to 1, the wind value is set to $0\,\mathrm{m\,s^{-1}}$.

4 Rain sensor

The Rain Sensor operates in the range 0 mm to 20 mm (0 V→3.3 V).It produces as output an analog voltage linearly proportional to the sensed rain level and it is sampled every 60 seconds.

4.1 Implementation

I chose pin PTB2 as analog input with the ADC0 to make the digital conversion. The task in charge of this requirement waits for a semaphore until the ADC0 interrupt handler provides a new data, then computes it and updates the specific variable.

5 Thermometer

The thermometer operates in the range $-20\,°C$ to $60\,°C$ ($0\,V \rightarrow 3.3\,V$). It produces as output an analog voltage linearly proportional to the temperature and it is sampled every 60 seconds.

5.1 Implementation

I chose pin PTB10 as analog input with the ADC1 to make the digital conversion. The solution is the same of Rain sensor.

6 Earthquakes detector

The Weather Station uses the accelerometer available on the FRDM K64F to detect earthquakes and transmits the data through the serial port along with the other sensed data.

6.1 Implementation

The accelometer is configured to sample every second the X,Y,Z axes and update the respective variables. Some steps are performed:

- **Initialization of the MK64F12 MCU(MCU_Init())**: Turn on the clock to I2C0 module and enable it.
- **I2C data write and read operations**: set the 7-bit I2C address of the FXOS8700CQ is 0x1D in its header and edit some macros in the I2C.h for the I2C0 module.
- **Initialization of the FXOS8700CQ(FXOS8700CQ_Init ())**: At the beginning of the initialization, all registers are reset to their default values by setting the RST bit of the CTRL_REG2 register. Then the FXOS8700CQ is initialized writing some registers.
- **Simple accelerometer calibration(FXOS8700CQ_Accel_Calibration ())**: compute axes acceleration values and perform an offset correction.
- **Accelerometer task(Accelerometer())**: it is the task that every second update the axes variable with the new values.

7 Conclusion

I spend a lot of time for debugging and trying different configurations, but it was a very interesting experience in which I improved my hardware and software programming skills with a real application and a physical board. I am very satisfied, thank you for giving us this opportunity.