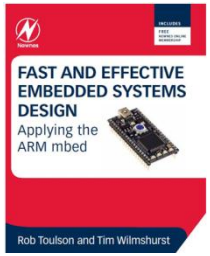
Embedded Systems Design Course

Applying the mbed microcontroller

Timers and interrupts

These course notes are written by R.Toulson (Anglia Ruskin University) and T.Wilmshurst (University of Derby). (c) ARM 2012

These course notes accompany the textbook “Fast and effective embedded system design : Applying the ARM mbed”



Timers and interrupts

- Time and event management in embedded systems
- An introduction to timers
- Using the mbed Timer object
- Using multiple timers
- Using the mbed Ticker object
- Hardware interrupts
- External interrupts on the mbed
- Switch debouncing for interrupt control
- Extended exercises

Time and event management in embedded systems

- Many embedded systems need high precision timing control and the ability to respond urgently to critical requests
- For example:
  - A video camera needs to capture image data at very specific time intervals, and to a high degree of accuracy, to enable smooth playback
  - A automotive system needs to be able to respond rapidly to a crash detection sensor in order to activate the passenger airbag
- Interrupts allow software processes to be halted while another, higher priority section of software executes
- Interrupt routines can be programmed to execute on timed events or by events that occur externally in hardware
- Routines executed by events that occur from an external source (e.g. a mouse click or input from another program) can be referred to as 'event driven'.

An introduction to timers

- Interrupts in embedded systems can be thought of as functions which are called by specific events rather than directly in code.
- The simplest type of interrupt is one which automatically increments a counter at a periodic interval, this is done behind the scenes while the software is operating.
- Most microcontrollers have built in timers or real-time-interrupts which can be used for this purpose.
- The main code can then be executed at specified time increments by evaluating the counter value.
- For example, we can set some pieces of software to operate every 10ms and others to operate every 100ms. We call this scheduled programming.

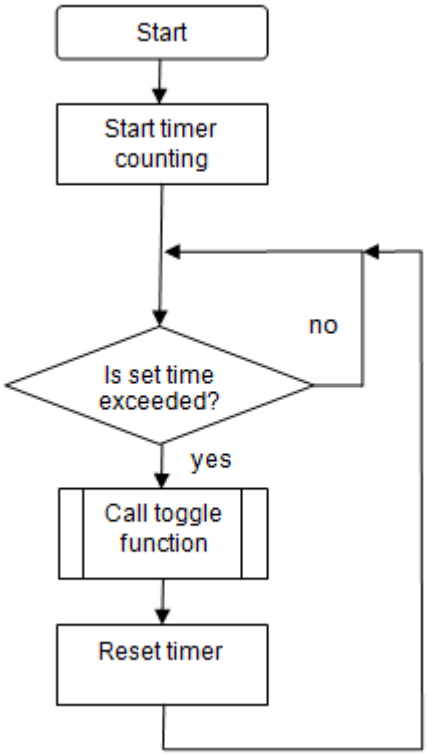
Using the mbed Timer object

We can use the mbed Timer object to perform scheduled programming:

Timer	**A general purpose timer**
Functions	**Usage**
start	Start the timer
stop	Stop the timer
reset	Reset the timer to 0
read	Get the time passed in seconds
read_ms	Get the time passed in mili-seconds
read_us	Get the time passed in micro-seconds

A simple timer routine

- Exercise 1: Create a square wave output using scheduled programming and verify the timing accuracy with an oscilloscope.




```
#include "mbed.h"

Timer timer1;            // define timer object
DigitalOut output1(p5); // digital output
void task1(void);        // task function prototype

//*** main code
int main() {
    timer1.start();      // start timer counting
    while(1) {
        if (timer1.read_ms()>=200)    // read time in ms
        {
            task1();                  // call task function
            timer1.reset();           // reset timer
        }
    }
}

void task1(void){           // task function
  output1=!output1;         // toggle output
}
```

Using multiple timers

- With scheduled programs we often need to execute different sections of code at different rates.
- Consider an automotive system:
  - The engine spark, valve and fuel injection system needs to be controlled and executed at a high speed, perhaps every 1 ms or less given that the engine revolves at anything up to 8,000 revs per minute.
  - The fuel tank level monitoring system needs to report the fuel level less often, perhaps every 1000 ms is sufficient.
- There is no point in executing both the injection management and the fuel level management systems at the same rate.
- For this reason we can use synchronous programs to improve efficiency.

Using multiple timers

- Exercise 2: Add a second timer which will run at a different rate, you can use an LED or an oscilloscope on the mbed pins to check that the two timers are executing correctly.

```
//*** main code
int main() {
    timer1.start();     // start timer1 counting
    timer2.start();     // start timer2 counting
    while(1) {
        if (timer1.read_ms()>=200)   // read time
        {
            task1();          // call task1 function
            timer1.reset();          // reset timer
        }
        if (timer2.read_ms()>=1000)  // read time
        {
            task2();          // call task2 function
            timer2.reset();          // reset timer
        }
    }
}

// continued...
```

```
// ...continued

//*** task functions
void task1(void){
  output1=!output1;     // toggle output1
}

void task2(void){
  output2=!output2;     // toggle output2
}
```

- Note: You will need to define a second timer object, digital output and task function prototype.

Challenges with timer interrupts

- With scheduled programming, we need to be careful with the amount of code and how long it takes to execute.
- For example, if we need to run a task every 1 ms, that task must take less than 1 ms second to execute, otherwise the timing would overrun and the system would go out of sync.
  - How much code there is will determine how fast the processor clock needs to be.
  - We sometimes need to prioritize the tasks, does a 1ms task run before a 100ms task? (because after 100ms, both will want to run at the same time).
  - This also means that pause, wait or delays (i.e. timing control by 'polling') cannot be used within scheduled program designs.

Using the mbed Ticker object

- The Ticker interface is used to setup a recurring interrupt to repeatedly call a designated function at a specified rate.
- Previously we used the Timer object, which required the main code to continuously analyse the timer to determine whether it was the right time to execute a specified function.
- An advantage of the ticker object is that we don't need to read the time, so we can execute other code while the ticker is running in the background and calling the attached function as necessary.

Using the mbed Ticker object

The mbed ticker object can also be used for scheduled programming.

Ticker	**A Ticker is used to call a function at a recurring interval**
Functions	**Usage**
attach	Attach a function to be called by the Ticker, specifying the interval in seconds
attach	Attach a member function to be called by the Ticker, specifying the interval in seconds
attach_us	Attach a function to be called by the Ticker, specifying the interval in micro-seconds
attach_us	Attach a member function to be called by the Ticker, specifying the interval in micro-seconds
detach	Detach the function

Using the mbed Ticker object

- Exercise 3: Use two tickers to create square wave outputs.
- Use an LED or an oscilloscope on the mbed pins to check that the two tickers are executing correctly.

```
#include "mbed.h"
Ticker flipper1;
Ticker flipper2;
DigitalOut led1(p5);
DigitalOut led2(p6);

void flip1() {                          // flip 1 function
    led1 = !led1;
}
void flip2() {                          // flip 2 function
    led2 = !led2;
}

int main() {
    led1 = 0;
    led2 = 0;

    flipper1.attach(&flip1, 0.2); // the address of the
                                  // function to be attached
                                  // and the interval (sec)
    flipper2.attach(&flip2, 1.0);

    // spin in a main loop
    // flipper will interrupt it to call flip

    while(1) {
        wait(0.2);
    }
}
```

Hardware interrupts

- Microprocessors can be set up to perform specific tasks when hardware events are incident.
- This allows the main code to run and perform its tasks, and only jump to certain subroutines or functions when something physical happens.
  - i.e. a switch is pressed or a signal input changes state.
- Interrupts are used to ensure adequate service response times in processing.
- The only real disadvantage of interrupt systems is the fact that programming and code structures are more detailed and complex.

External interrupts on the mbed

External interrupts on the mbed:

InterruptIn	A digital interrupt input, used to call a function on a rising or falling edge
Functions	**Usage**
InterruptIn	Create an InterruptIn connected to the specified pin
rise	Attach a function to call when a rising edge occurs on the input
rise	Attach a member function to call when a rising edge occurs on the input
fall	Attach a function to call when a falling edge occurs on the input
fall	Attach a member function to call when a falling edge occurs on the input
mode	Set the input pin mode

- Note: any digital input can be an interrupt except pin 19 and pin 20

External interrupts on the mbed

- Exercise 4: Use the mbed InterruptIn library to toggle an LED whenever a digital pushbutton input goes high.

```
#include "mbed.h"

InterruptIn button(p18);     // Interrupt on digital pushbutton input p18
DigitalOut led1(p5);         // digital out to p5

void toggle(void);           // function prototype

int main() {
    button.rise(&toggle);    // attach the address of the toggle
}                                        // function to the rising edge

void toggle() {
    led1=!led1;
}
```

- You may notice some issues with this simple program, what are they?

Switch debouncing for interrupt control

- Exercise 4 doesn't work quite as expected; it is possible for the button to become unresponsive or out of synch with the LED.
- This is because of a common issue called switch or button bouncing. When the button is pressed it doesn't cleanly switch from low to high, there is some 'bounce' in between as shown below:

- It is therefore easy to see how a single button press can cause multiple interrupts and hence the LED can get out of synch with the button.
- We therefore need to 'debounce' the switch with a timer feature.

Switch debouncing for interrupt control

- Exercise 5: Use the mbed InterruptIn library to toggle an LED whenever a digital input goes high, implementing a debounce counter to avoid multiple interrupts.

```
#include "mbed.h"

InterruptIn button(p18);     // Interrupt on digital pushbutton input p18
DigitalOut led1(p5);         // digital out to p5
Timer debounce;              // define debounce timer

void toggle(void);           // function prototype

int main() {
    debounce.start();
    button.rise(&toggle);       // attach the address of the toggle
}                                           // function to the rising edge

void toggle() {
if (debounce.read_ms()>200)     // only allow toggle if debounce timer
    led1=!led1;                             // has passed 200 ms
    debounce.reset();           // restart timer when the toggle is performed
}
```

Switch debouncing for interrupt control

- By removing the bounces, the system acts as though the switch has an ideal response.
- This can be done by a variety of hardware and software methods.

- An example of a classic hardware debouncer would be two cross-coupled NAND gates form a very simple Set-Reset (SR) latch.
- Another example of a software debouncer would be to look for a number of sequential readings of the switch, e.g. if the input changes from 0 to 1 and then continues to read 1 for the next ten samples then the switch has been pressed.

Extended exercises

- Exercise 6: Using an oscilloscope evaluate the debounce characteristic of your pushbutton. What is the ideal debounce time for your pushbutton? Note that longer debounce times reduce the capability for fast switching, so if fast switching is required a different type of pushbutton might be the only solution

- Exercise 7: Combine the timer and hardware interrupt programs to show that a scheduled program and an event driven program can operate together. Flash two LEDs at different rates but allow a hardware interrupt to sound a buzzer if a pushbutton is pressed.

- Exercise 8: Accelerometer chips such as the ADXL345 have interrupt output flags to enable an interrupt based on an excessive acceleration (as used in vehicle airbag systems). Investigate and experiment with the ADXL345 interrupt feature to sound a buzzer when a high impact is seen.

Summary

- Time and event management in embedded systems
- An introduction to timers
- Using the mbed Timer object
- Using multiple timers
- Using the mbed Ticker object
- Hardware interrupts
- External interrupts on the mbed
- Switch debouncing for interrupt control
- Extended exercises