Lab 3: 3D Tilt Angle and Tap Detection using Accelerometer

Contents

Abstract

This project involves the construction of a system that calculates the STM32F4DISCOVERY board's tilt angles in 3 dimensions with 4 degree accuracy. The system is also able to detect a single tap and turns on/off the 4 user LEDs in response to the tap. The accelerometer is calibrated to operate on any inclined surface, and the data is filtered using a moving average filter before being displayed on the 4 LEDs. A window depth of 50 was found to be best for the moving average filter. Our device accurately responded to 8 out of 10 taps. Additionally, every angle measurement was also within the allowable 4º range of accuracy.

1 Problem Statement

We were required to develop a program with two major functions: angle measuring and tap detection.

1.1 Angle measuring

Using the acceleration sensors, we were required to determine the current pitch and roll of the discovery board. As explained in the theory section, it isn't possible to measure the yaw while restricting oneself to the accelerometer sensor values. The accelerometer sensors had to be calibrated off-line and during program execution, their values were to be piped through a moving average filter. The error in the calculated angles wasn't allowed to exceed 4 [degrees]. These angles had to be measured and printed to the debug screen at least 100 times per second. A hardware timer (TIM3) was to be responsible for ensuring this timing. It had to be set to the desired frequency (over 100Hz) and was to trigger an interrupt which starts the angle computation.

1.2 Tap detection

The accelerometer has built in capability to detect "taps". A tap is when the device reads a sharp and transient increase in acceleration. This can be achieved by tapping the device with one's finger. We were required to configure the accelerometer to trigger interrupts upon detecting a tap. The state of the LEDs (Off or On) was to be toggled when the microprocessor receives this interrupt This interrupt was to have a lower priority than the TIM3 interrupt All LEDs shared the same state so a tap would turn them all on, and a subsequent tap would turn them all off.

> Students should not be tapping the board like 10 times in order to make it respond
>
> — Anonymous *Microprocessors course news feed*

2 Theory and Hypothesis

2.1 Theory

Looking at the theory behind the following systems: the accelerometer, the acceleration to angle conversion, the timer, the Nested Vector Interrupt Controller and the External Interrupt.

2.1.1 Accelerometer

The LIS302DL is a 3-axis linear accelerometer. It returns a digital value representing static forces caused by a constant force like gravity, or dynamic ones like moving or vibrating the accelerometer. The accelerometer includes a sensing element and an IC interface that provides accelerations through I2C/SPI serial interface. The stm32f4_discovery_lis302dl library is used to configure the parameters of the device.

The LIS302DL is capable of measuring output data rates at 100 and 400 Hz and has dynamically user selectable full scales that have the following typical values: $\pm$ 2.3g and $\pm$9.2g. The user can select which axis he wants to read the acceleration from. Additionally, a self test capability allows the user to check the functioning of the sensor in the final application.

The device may be configured to generate interrupt signals when a programmable threshold is crossed in at least one of the three axes. The user can select between a latched and unlatched interrupt request selection. A time limit can be set to determine how long a threshold is sensed before an interrupt is signalled. Additionally, an option is provided to enable single or double tap detection in each direction.

2.1.2 Acceleration to Angle Conversion

Tilt angles are used to describe how the accelerometer is oriented in 3D space relative to a global XYZ frame. The XY plane is horizontal to the earth, and the Z plane is perpendicular to it. When rotated around this frame, the accelerometer changes angles with respect to the coordinate system. We will refer to these angles as roll (between the X axis and the horizontal plane), pitch (between the Y axis and the horizontal plane) and yaw (with respect to the Z axis). Figure 1 shows these angles and Figure 2 shows our chosen positive XYZ direction on the STM32F4DISCOVERY board. Note that yaw cannot be detected because gravity force does not depend on it[1].

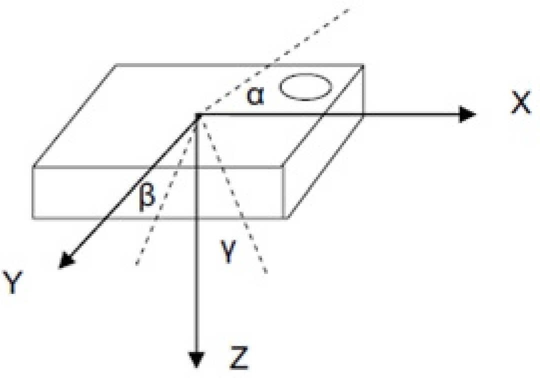

Figure 1: Yaw (beta), Pitch (gamma) and Roll (alpha)

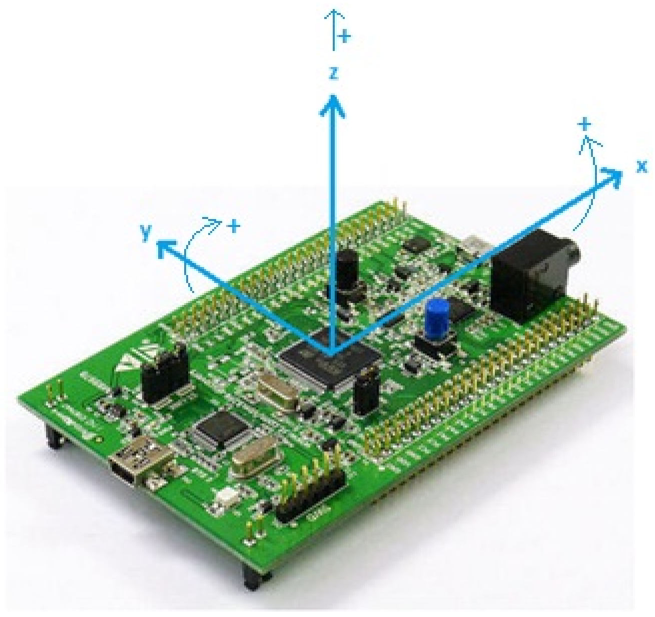

Figure 2: Positive roll, yaw, and pitch on our board

By measuring the static acceleration due to gravity, we can find out the angle of the device. Equations 1 and 2 [4] allow us to convert the acceleration detected in each direction into a tilt angle.

Equation 1

$$\alpha = \arctan\left(\frac{\alpha_X}{\sqrt{\alpha_Y^2+\alpha_Z^2}}\right)$$

Equation 2

$$\alpha = \arctan\left(\frac{\alpha_Y}{\sqrt{\alpha_X^2+\alpha_Z^2}}\right)$$

Equations 1 and 2 return angles between -90 and 90 degrees. Figure 2 shows how a range of -180 to 180 can be obtained. Depending on the direction of the board, the angles obtained from Equations 1 and 2 can be used in Equations 3 to 6 to modify the range.

Equation 3

$if(z < 0 and x < 0) : roll = -(roll + 180)$

Equation 4

$if(z < 0 and x > 0) : roll = 180 - roll$

Equation 5

$if(z < 0 and y < 0) : roll = -(pitch + 180)$

Equation 6

$if(z < 0 and x > 0) : roll = 180 - pitch$

2.1.3 The Nested Vector Interrupt Controller

The Nested Vector Interrupt Controller (NVIC) provides configurable interrupt handling abilities to the processor. Whenever an interrupt is signalled, the NVIC saves the program state to the stack and begins execution of a handler indicated in a vector table at the beginning of flash memory.

The NVIC has a total of 240 user configurable interrupts, each with up to 256 levels of priority. The priority can be changed dynamically. The NVIC maintains knowledge of the stacked or nested interrupts to enable tail-chaining of interrupts.

Through the NVIC, we can specify the IRQ channel to enable or disable, as well as the pre-emption priority for that channel. This parameter can be a value between 0 and 15, where a lower priority value indicates a higher priority. We can also select a sub-priority for the interrupts with the same characteristics as the pre-emption priority parameter. [5]

2.1.4 Timer

4 general purpose timers are available and consist of auto-reload counters driven by a programmable prescaler (TIM2 to TIM5). TIM3 is a 16-bit auto-reload counter and is used to control the interval of the accelerometer readings.

Functions to access the timer settings are available in the stm32Fxx library. The timer has a parameter which allows it to be an up or a down counter. A down counter was used in this lab meaning that the timer will generate an event upon hitting zero and will then reload an initial value. Every time this happens, an interrupt occurs. A 16 bit programmable prescaler is used to divide the counter clock frequency to reduce the tick rate of the timer[6].

2.1.5 External Interrupts

The external interrupt (EXTI) is a configurable edge-triggered interrupt on GPIO pins. It consists of up to 23 edge detectors.

Each input line can be configured to select GPIO port to be used as a source for the EXTI line. The user can also specify which EXTI lines should be enabled or disabled as well as the mode for the exit line (interrupt or event). We can specify the trigger signal active edge between rising, falling, or rising-falling[7].

2.2 Hypothesis

The external interrupt (EXTI) is a configurable edge-triggered interrupt on GPIO pins. It consists of up to 23 edge detectors. Each input line can be configured to select GPIO port to be used as a source for the EXTI line. The user can also specify which EXTI lines should be enabled or disabled as well as the mode for the exit line (interrupt or event). We can specify the trigger signal active edge between rising, falling, or rising-falling.

3 Implementation

The required structure of this lab shared much in common with lab 2 so we decided to use that as a starting point. After modifying the main program loop from lab 2 to suit our needs, we worked on configuring the peripherals and reading from them. We also had to enable interrupt functionality for the timer and the accelerometer. Once we had this working, we calibrated the accelerometer and then determined the ideal window depth for the moving average filter.

3.1 Program Flow

The program flow consists of three threads. Here they are ordered in descending execution priority. -The timer inerrupt handler -The tap interrupt handler -The main thread

The default thread is the main thread; when the interrupt threads have nothing to do, the CPU executes the main thread. Minimal time is spent in the interrupt handlers and they only set a flag and then clear their respective interrupts when called. The main thread looks for changes in these flags and reacts accordingly. The program flow and the interaction between the processes is summarized in Figure 3.

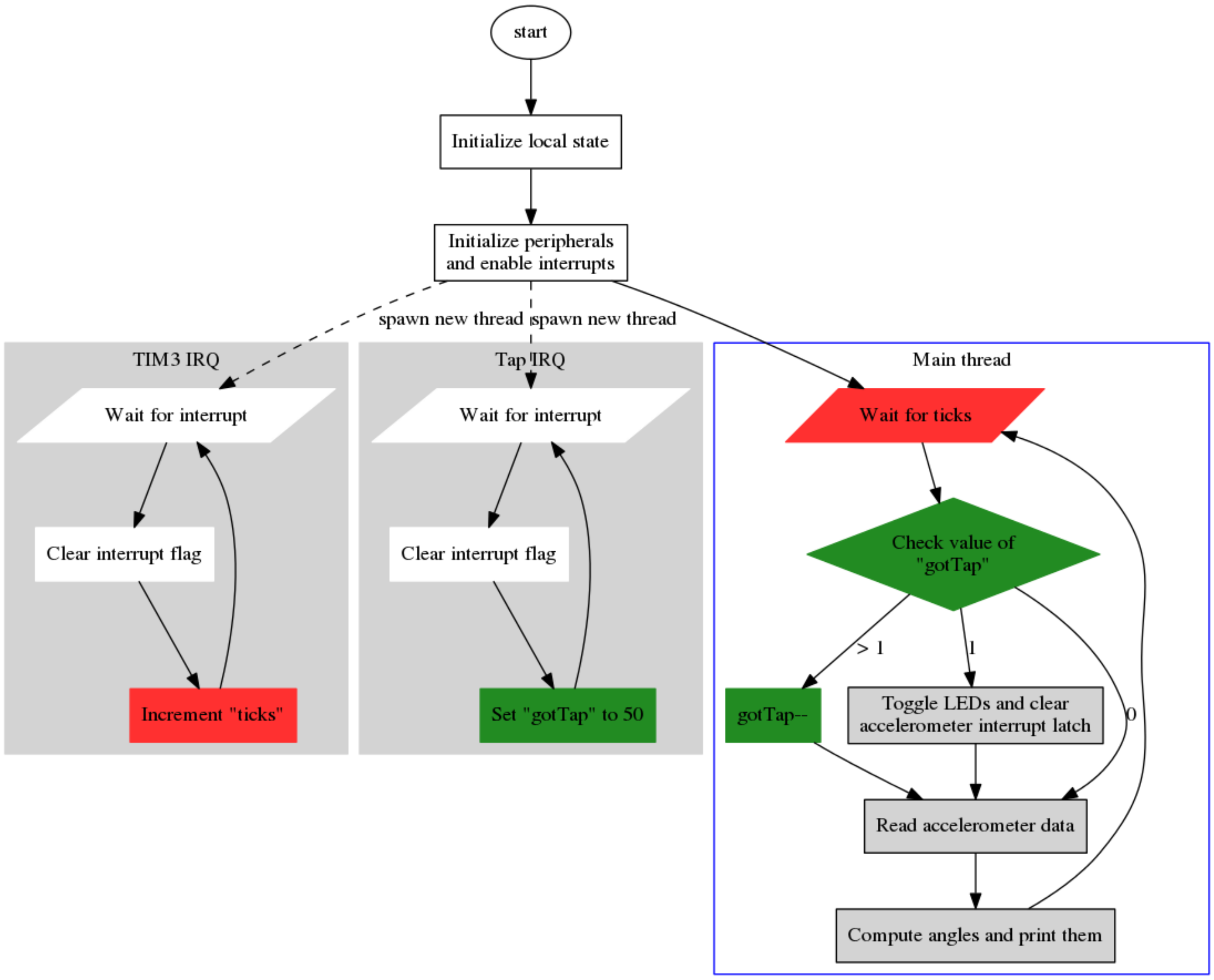

Figure 3: Top level flowchart

It should be noted that this program isn't thread safe. Making the program thread safe while respecting the requirements (keep interrupt handlers fast) wasn't feasible without temporarily disabling interrupts (which would defeating the purpose of having them) or using some form of mutex. Because of the nature of the program flow, it can be shown that the program will always get back on it's feet even though there might be a collision. We decided to accept the occasional glitch in the name of simplicity. In Figure 3, accesses to shared resources can be seen as colored nodes.

An alternative to our design would be to have the main thread wait for either the timer flag or the tap flag instead of just waiting for the timer flag and checking for the tap flag then. This modification to the main thread can be seen in Figure 4.

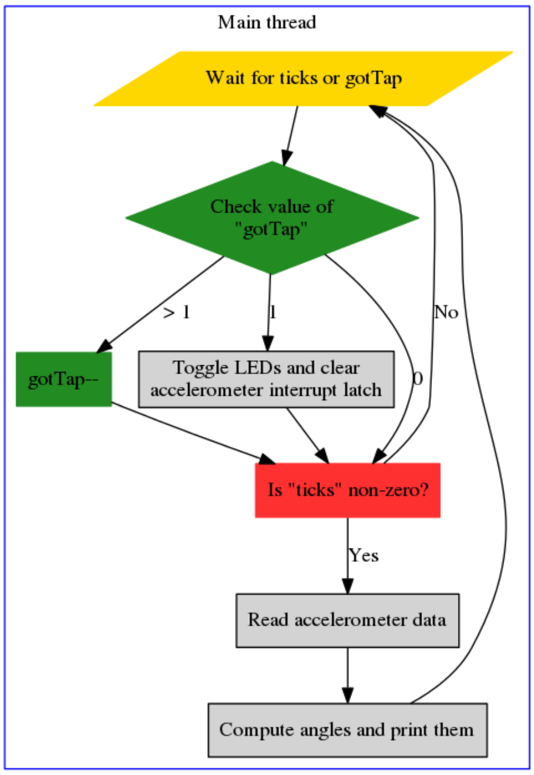

Figure 4: Modified main thread

This program flow would be more in the spirit of the interrupt driven program described in the lab, but we opted out of doing that as it introduced additional potential collisions.

3.1.1 Filter depth

We determined in lab 2 that, given enough memory resources, the larger the depth the better. The upper bound being the response time of the filter; response time meaning how long it takes for the fitler to output to converge given fixed input. We decided that a response time of about 0.5 seconds was good for measuring angles. Since the angles were being computed at 100Hz, a filter depth of 50 would give us the required response time.

3.2 Interrupt and peripheral configuration

Each of the following system/peripheral was initialized with specific parameters in order to achieve tap detection and acceleration reading: the accelerometer, the NVIC, the timer, and the External Interrupt.

3.2.1 The accelerometer

The accelerometer's power mode is set to active in order to turn on the device. The data rates at which acceleration samples can be produced are 100Hz and 400 Hz. For the purpose of this lab, a 100 Hz data rate was chosen. Because this lab requires a sampling of 100 Hz through the timer, it is unnecessary to consume additional power for a more frequent sampling rate. All axes were configured since all 3 accelerations are needed to determine the tilt angles. The LIS302DL has dynamically user selectable full scale measurement range of +-2.3g and +-9.2g. We chose a +-2.3g range since we are only concerned with static acceleration. Moreover, a lower range is associated with more accurate readings. A self-test capability allows the user to check the functioning of the sensor without moving it. In this lab, we turn off this functionality since it isn't necessary.

3.2.2 Tap Detection

We configure the device to generate interrupt signals with specific parameters. We selected an unlatched interrupt request so that the interrupt doesn't send us multiple signals before we get a chance to react to the first one. A single click interrupt was enabled in the Z direction because we are interested in detecting tapping in that direction alone. In addition to the automatic setting of the appropriate bits in the click configuration register, we set the first three bits of the interrupt control register as well. This was necessary to enable interrupts in the Z direction. Double clicking is disabled for the purpose of this lab. We selected the maximum value for the threshold of the tapping detection to decrease the sensitivity of the tapping. Lastly, the timing limit was set to a minimum to ensure that once a tap goes past the threshold, it is immediately detected.

3.2.3 Calibration

Because the sensor readings aren't perfect, calibration is required to correct these deviations. In this experiment, we apply an off-line calibration technique where the process of calibrating the parameters is done only once (not every time the board is powered on). Calibration is done by putting the board in three different directions and taking a sequence of measurements in each direction.

We start by placing the board in the direction orthogonal to the horizontal plane and take a sequence of measurements in that direction. The offset is the value which centers the reading in each direction. After applying the offset, we get two values centered around zero, but with non-unity magnitude. A scalar multiplier is used to correct for this. We call this value the scalar multiplier. This calibration method is used for every direction, and stored to reapply to each sample.

Because this was an offline calibration, we put the constants as `#define` s in the accelerometer file and internally passed the values through a private calibration function which used these constants whenever a reading was taken from the accelerometer.

We could have instead passed the calibration constants to the accelerometer module at initialization, but then they would have simply been `#define` s in another file and the program would suffer a speed penalty for no added functionality in our use case.

3.2.4 NVIC

The NVIC is used to set the priority of the interrupts. We chose the timer to have the higher priority because we want to ensure that we sample at regular intervals. If the tap interrupt has the higher priority, it will disrupt the regular sampling of the accelerometer.

3.2.5 EXTI

The SYSCFG clock must be enabled before connecting the output of the accelerometer to the EXTI line. We selected the first EXTI line (EXTI_Line0) to be attached to the sensor, and set the interrupt mode instead of the event mode. This is done because we would like the processor to run the handler immediately instead of having a queued event.

The EXTI_Line0 is checked to determine whether an interrupt has occurred. When an interrupt occurs, the EXTI's line pending flag is cleared so that the system is ready to wait for the triggering of another interrupt.

3.2.6 TIMER

The timer has its own clock that must be enabled before being used (the APB1 clock). The structure allows us to choose a prescaler value (integer clock divider to reduce the tick rate of the timer) as well as a reload value (timer generates an event upon hitting zero and reloads some initial value). The period was calculated by dividing the APB1 clock by a prescaler and the desired frequency of the timer as seen in Equation 7.

Equation 7

$$Period = \left(\frac{f_P CLK1}{Prescaler * frequency}\right) - 1$$

The prescaler and period specified as parameters must be between 0x0000 and 0xFFFF. Because of this, we chose our prescaler to be 42,000. The APB1 clock operates at a frequency of about 42Mhz giving us a value of 1000 for the counter. This means that one second is equivalent to when the counter counts down from 1000 to zero. We then divide 1000 by the desired frequency.

The frequency specified for this lab is 100 Hz, so dividing 1000 by 100 gives us 10. This means that the counter must count down to one hundredth of a 1000at each tick. We finally subtract 1 from this value because of the off by 1 issue. We want to get n decrements rather than n+1 seeing that the timer must count all the way down to zero. The TIM update Interrupt source is checked to determine when the down counter reaches zero. If the bit status is set (meaning an interrupt has occurred), the interrupt pending bit is cleared so that the system is ready to wait for the triggering of another interrupt rather than staying in the interrupted state.

4 Testing and Observations

To test the system, we ran the program on the board while trying to throw all the corner cases at it. Because of the system's simplicity, there weren't many corner cases. We tapped and shook the device many times and made sure it was still behaving properly.

4.1 Calibration

Once we had the program up and running, we needed to compute the calibration constants. We "short circuited" the print statement to print out filtered accelerometer values instead of angles. We set the filter depth to an arbitrary large number (we chose 300) and waited for the filter value to converge while holding it in each of the three positions that lined up the direction base vectors with gravity. Having a large filter depth gave us an average of the accelerometer readings. This saved us the trouble of computing the average of the raw data using another method. We then used the method described in the theory section to compute the calibration constants.

4.2 Timer

We set the timer period to trigger interrupts at a frequency of 1Hz. This was so we could validate our period equation as validating a frequency of 100Hz required very fast fingers on the stop watch. We got some strange results and our timer was initially going four times too fast. We were aware of various problems groups had with the timer, but even after looking at the documentation and finding the relevant clocks and referencing their values from the libraries directly, our timer was too fast by a factor of 1.5. After hours of trials and sifting through documentation, we decided to put a fudge factor in our equation and leave it at that. Once this was fixed, we got a very stable interrupt frequency. We tested the interrupt frequency at 100 Hz by incrementing a counter at each interrupt and measuring the time elapsed after 1000 interrupts. This should have taken 10 seconds and we measured 10.2.

Another way to test the running time of the timer was to break in the code and see how much time has elapsed. This method gave us garbage values. We suspect that the TIM3 timer being an external hardware timer made it not play nice with the debugger.

4.3 Tap detection

The tap detection was initially way too sensitive: a single tap would cause the LEDs the flicker as the board received many interrupts. This occurred even at the lowest sensitivity. After making the interrupt latch, we got better behavior, but a single tap still often triggered two interrupts causing the LEDs to quickly toggle state twice. Our last modification, adding debouncing to the main program flow, caused much better behavior.

On average, our device accurately responded to 8 out of 10 taps (an inaccurate response being either the LEDs toggling then returning to their initial state of no response at all). The way the device was tapped affected the accuracy: gently taps yielded better success rates. Table 1 shows the success rate versus "tap strength".

Table 1: Tap detection success

Hard	Medium	Gentle
6/10	9/10	9/10

5 Conclusion

This project involved the construction of a system that calculates the board's tilt angles in 3 dimensions with 4 degree accuracy. The system is also able to detect a single tap and turns on/off the 4 user LEDs in response to the tap. The device was calibrated and the data run through a moving average filter. Since the angles were being computed at 100Hz, a filter depth of 50 gave us the required response time. On average, our device accurately responded to 8 out of 10, but the way the device was tapped affected the accuracy: softer taps yielded better success rates. Lastly, every angle measurement was also within the allowable 4º range of accuracy.

REFERENCES

[1] LIS302DL. *MEMS motion sensor 3-axis - ± 2g/± 8g smart digital output "piccolo" accelerometer* . Page 1.

[2] AN3182 Application Note. *Tilt measurement using a low-g 3-axis accelerometer* . Page 13, Figure 8.

[3] http://www.embedds.com/debugging-stm32-discovery-under-linux/ Date last accessed: Feb.-25-13

[4] AN3182 Application Note. *Tilt measurement using a low-g 3-axis accelerometer*. Page 13, Equation 6 and 7.

[5] Cortex-M4 Revisionr0p1. *Technical Reference Manual*. Sections 6.1 – 6.2.

[6] RM0090 Reference manual. *STM32F405xx, STM32F407xx, STM32F415xx and STM32F417xx advanced ARM-based 32-bit MCUs*. Section 14.

[7] RM0090 Reference manual. *STM32F405xx, STM32F407xx, STM32F415xx and STM32F417xx advanced ARM-based 32-bit MCUs*. Section 9.