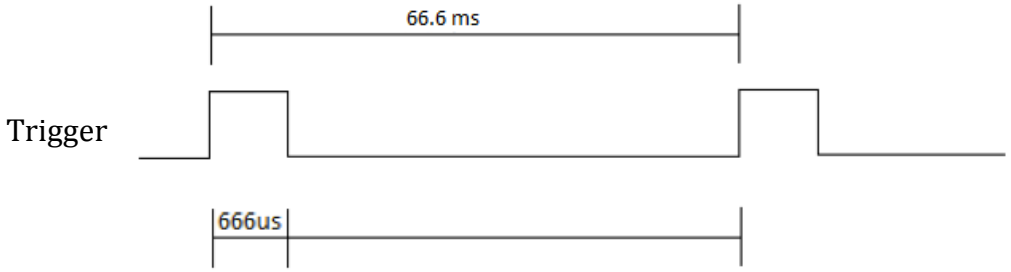
Operating Systems for Embedded Systems
Assignment 1: proximity alert system

Student: Palmiero Christian (ID: 231599)

1. Software architecture

The software starts from the main() function, which initializes the hardware and the GPIO pins, installs the interrupt handler routine for the IRQ line of the third instance of the FlexTimerModule (FTM3), creates a semaphore with an initial value equal to 0 and a mutex sempahore, creates the start task, called 'AppTaskStart', and gives the control to uc/OS-III.
The 'AppTaskStart' task initializes the uC/CPU services, the memory management module, the mathematical module and a 115200 baud rate serial port; then, it creates three tasks: 'Trigtask', 'MainTask' and 'BlinkingTask'.
The 'TrigTask' task generates the Trigger input signal of the HC-SR04 proximity sensor by using the instance 0 of the Flex Timer Module, configured in edge-aligned PWM mode. This configuration allows to produce a square wave with fixed period and duty cycle. The period is 1/(15 Hz) = 66.6 ms and the duty cycle is 1%: in this way, the positive pulse of the trigger signal in a single period lasts 666 μs (see the picture below). Consecutive activations of the HC-SR04 sensor using the Trigger input must be delayed by at least 60ms and a positive pulse of width equal to at least 10μs must be applied to the Trigger input. These requirements are fulfilled with the given parameters.



The FTM0 channel 0 is assigned to the pin 1 of the PORTC, with the 'Alt' functionality set to 4. Afterwards, the 'FTM_DRV_INIT' function enables the clock for the current timer instance, resets the FTM0 registers, sets the prescaler factor to 128, uses the Enhanced PWM synchronization method, sets the mode of the WPDIS protection bit to 1 and clears the pending bit of an external trigger. The 'FTM_DRV_PwmStart' function, instead, sets the initial value of the counter (CNTIN) to 0, updates the MOD and the Channel0Value registers with their properly calculated values and selects as clock source the system clock (120MHz) in order to start the counter. In particular,

$$Channel0Value = \frac{width\ of\ the\ positive\ pulse * Prescaler\ factor}{Fck} + CNTIN$$

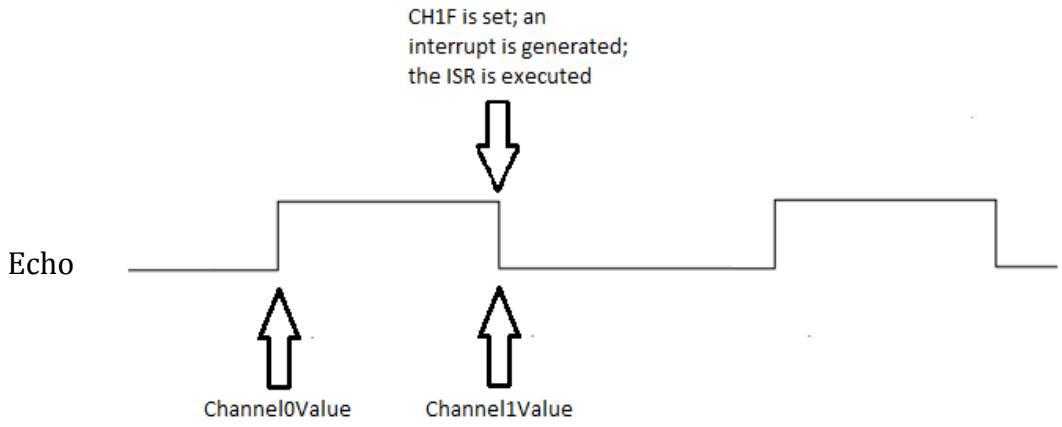
$$Mod = \frac{Period\ of\ the\ square\ wave * Prescaler\ factor}{Fck} - 1 + CNTIN$$

With a clock source of 120MHz and a prescaler factor of 128, the maximum period that can be generated is equal to 69.9 ms. The period of the Trigger signal is 66.6 ms; so, this requirements is also satisfied.
The 'MainTask' task allocates three vectors: 'first', 'last' and 'blinktime'. The first one contains, in each position, a value in centimetres that corresponds to the left boundary of a range of distances; the second one has the same purpose, but deals with the right boundary; the third one stores, in the i-th position, the blinking period in milliseconds corresponding to the specific range of distances defined by first[i] and last[i].

For example, if the distance of the nearest object is equal or greater than 2 meters and the GREEN led is required to blink with a period of 2.000 ms, first[i] is equal to 200 cm, last[i] to 401cm (4 meters is the maximum sensor-detectable distance), blinktime[i] to 2000 ms.
The 'MainTask' task handles the Echo output signal of the HC-SR04 proximity sensor by using the instance 3 of the Flex Timer Module, configured in dual-edge continuous capture mode with a clock source of 120MHz and a prescaler factor of 128. This configuration allows to measure the width of the positive pulse of the Echo signal and, from it, to compute the distance from the object's surface. The FTM3 channel 0, that hosts the Echo signal, is assigned to the pin 0 of the PORTD, with the 'Alt' functionality equal to 4. Afterwards, the timer is configured using several functions provided by the Kinetis SDK v1.0.0 (for further details, see code comments in the 'app.c' file and the Freescale "K64 Sub-Family Reference Manual").
The task body starts with an 'OSSemPend', that causes the task to wait on the 'Sem' semaphore. If the task is ready-to-run and is rescheduled, the distance is calculated according to the following formula:

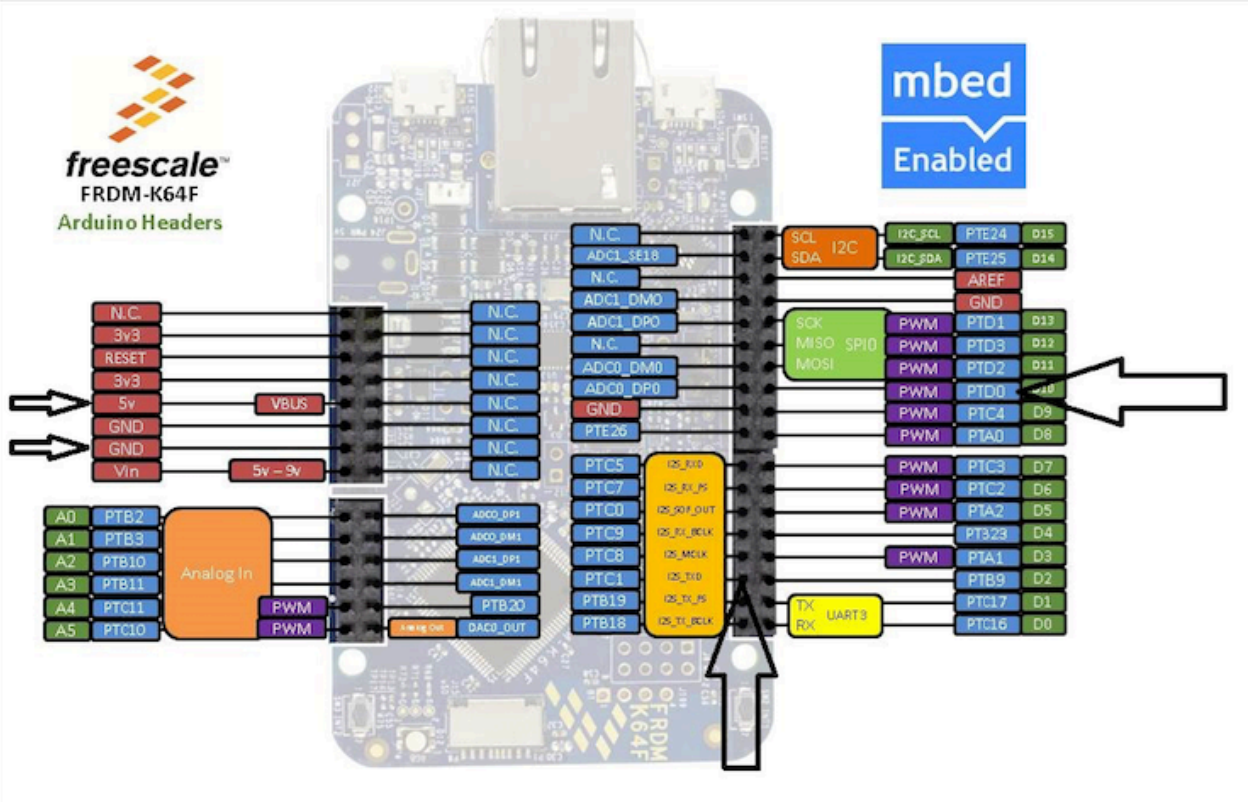
$$distance = \frac{(Channel1Value - Channel0Value) * PrescalerFactor * 10^6}{58 * Fck}$$

The difference between Channel1Value and Channel0Value is computed and stored in the global variable 'temp' by the interrupt service routine 'BSP_FTM3_int_hdlr' (further details on this function later on in this document). The task waits for two measurements, calculates and prints on the serial port their average in order to achieve a better accuracy, and computes the index corresponding to the range of distances the average belongs to. If the next blinking period is different from the current one, the latter is updated, the next LED that has to blink is assigned to the global variable 'led', all the LEDs are turned off and the 'BlinkingTask' task is resumed (if the task was not delayed, nothing happens). In this way, the system reacts quite swiftly to the new measurement and the right led immediately starts blinking with the new blinking period.
The portion of code that manipulates the global variables 'led', 'TableIndex', 'period' is surrounded by an 'OSMutexPend' and an 'OSMutexPost', so that shared resources are protected against concurrent accesses.
The 'BlinkingTask' task checks if at least one measurement has been computed and either toggles the right led or, in case the distance from the object is less than 10 cm, sets the red LED. Then, it is delayed by an amount of time that is equal to the blinking period. Even in this case, the mutex semaphore protects the shared resource 'led' and avoids that the task is interrupted while reading this resource, which can be also manipulated and modified by the 'MainTask' task.
The 'BlinkingTask' and the 'MainTask' can interact in different ways: the 'BlinkingTask' has been delayed, the 'MainTask' enters its critical section, resumes the 'BlinkingTask' and the blinking restarts correctly; the 'BlinkingTask' is immediately above or below the 'OSTimeDlyHMSM' function, is interrupted and the 'MainTask' enters its critical section. In the latter scenario, when the 'MainTask' exits from its critical section, the 'BlinkingTask' either performs the 'OSTimeDlyHMSM' with the new period, or executes its critical section without any problem.
Finally, the interrupt handler routine is executed when the FTM3 channel 1 interrupt is generated. If the CH1F bit is set, Channel0Value and Channel1Value are read, their difference is computed and stored in the global variable 'temp', CH0F and CH1F bits are cleared, the counter value is reset to 0 (CNTIN) and an 'OSSemPost' is performed, in order to make the 'MainTask' ready-to-run.
With a selected clock source of 120MHz and a prescaler factor of 128, the maximum period that can be measured is equal to 69.9 ms; it may happen that one overflow occurs between two falling edges of the Echo signal. If an overflow has occurred, the TOF bit is cleared and Channel0Value and Channel1Value are incremented by 0xFFFF, that is the final value of the counter (MOD).
The picture below points out how the Echo signal is handled by the software application.



2. Pin assignment

VCC->5v; GND->GND; Trig-> PTC1; Echo->PTD0



3. How to use the project

-Open Kinetis Design Studio 3.0.0 IDE and create a new folder to select as workspace
-File->Import->Existing projects into Workspace->Select archive file->ProximityAlarm.zip
-At this point, it may open a window that asks for the path of the IAR Embedded Workbench IDE installation directory; fill the empty field with the correct path and click on 'OK'
-Open the 'Project Explorer', click on 'OS3-KSDK' and then click on the hammer icon in order to build the project
-Connect the board to your PC via USB cable between the OpenSDA USB connector and the PC USB connector
-Open a terminal application, such as PuTTY or TeraTerm, connect to the debug serial port number and set the baudrate to 115200.
-Right click on the 'OS3-KSDK' project-> Debug as…-> Debug configuration
-Click on 'GDB SEGGER J-Link debugging' and then on 'OS3-KSDK Debug J-Link'
-Click on 'Debug' (Do not take care of the warning 'The connected emulator doesn't support SWO')
-Click on the 'Resume' icon