AN4705
Application note

Migration guidelines from PIC18 to STM32F0 Series with software expansion for STM32Cube

Introduction

This application note provides some guidelines and methodology to migrate easily from a Microchip PIC18 family based application to the STM32F0 Series platform. The migration is supported by the X-CUBE-PICTOF0 software package and includes the following main steps:

- How to select the appropriate sales type. Refer to the device selector available from the *www.st.com* website (search for X-CUBE-PICTOF0)
- How to use the tools to create a functional STM32F0 application, including:
  - the libraries selection
  - the integrated development environment (IDE)
  - the compiler.

The document contains also a short competitive analysis that highlights the advantages of such a migration, including the whole hardware and software ecosystem.

To fully benefit from this application note, the user should be familiar with the STM32 microcontrollers documentation available from *www.st.com* with a particular focus on:

- STM32F0 Series datasheets:
  - DS9773 (STM32F030xx)
  - DS10111 (STM32F031xx)
  - DS10249 (STM32F038xx)
  - DS10147 (STM32F042xx)
  - DS10213 (STM32F048xx)
  - DS8668 (STM32F051xx)
  - DS9145 (STM32F058xx)
  - DS10697 (STM32F070xx)
  - DS10009 (STM32F071xx)
  - DS9826 (STM32F072xx)
  - DS10212 (STM32F078xx)
  - DS10312 (STM32F091xx)
  - DS10624 (STM32F098xx)
- STM32F0 Series reference manuals (RM0091, RM0360)
- STM32F0xxx Cortex®-M0 programming manual (PM0215)
- Getting started with STM32F0x1/x2/x8 hardware development (AN4080)
- Getting started with STM32F030xx and STM32F070xx hardware development (AN4325)

Table 1. Applicable products

Type	Product series and part number
Microcontrollers	STM32F0 Series, X-CUBE-PICTOF0

Contents

List of tables

List of figures

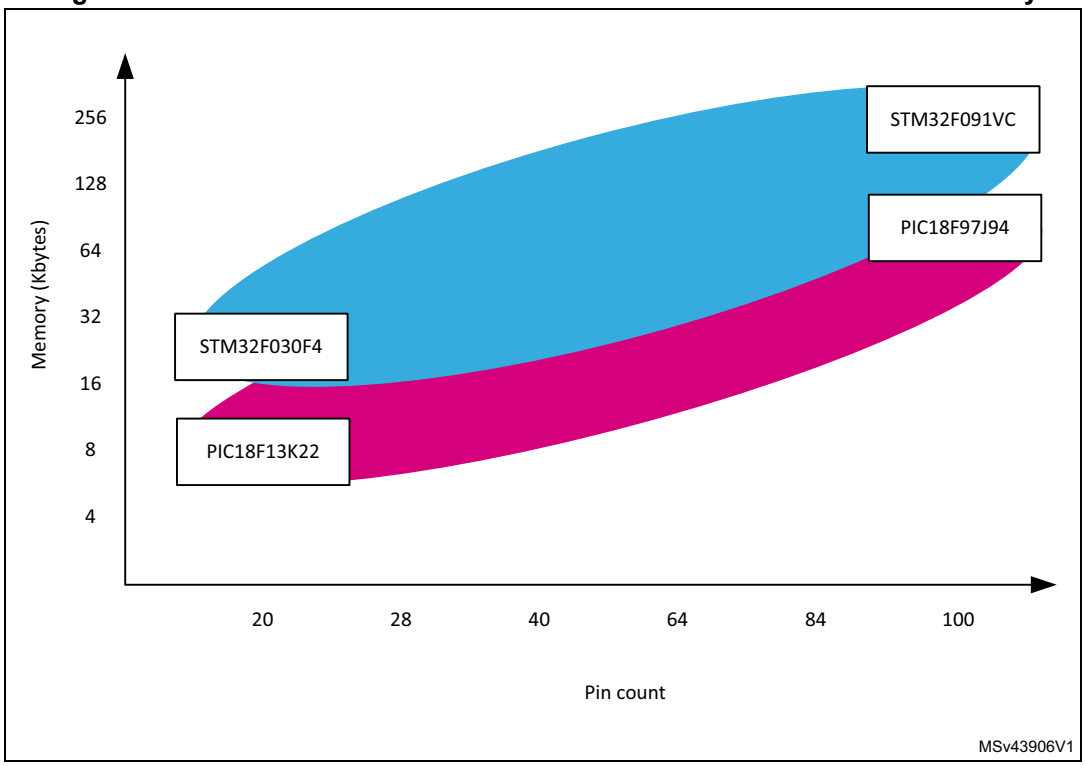
1 Portfolio comparison

The STM32 microcontrollers offer a competitive alternative to the PIC® (Programmable Intelligent Computer) family developed by the Microchip Technology company.

This application note describes how to secure the migration from the 8-bit platforms based on the PIC18 family to the 32-bit Cortex® MCU of the STM32F0 Series.

The Microchip PIC family includes a large number of different cores such as PIC12, PIC16, PIC18. The memory density (*Figure 1*) and the peripheral set of the PIC18 family are similar to the STM32F0 Series. Some differences exist on the core performance, on the bus architecture and on the peripheral capabilities.

Figure 1. Portfolio overview between the STM32F0 Series and the PIC18 family



1.1 PIC18 family

Microchip introduced the PIC18 family in year 2000, with an architecture delivering up to 16 MIPS of processing power. *Table 2* shows a more detailed overview of the PIC18 family and the STM32F0 Series.

Table 2. Comparison between the PIC18 family and the STM32F0 Series

	Microchip PIC18F series	STM32F0 Series
Processor	8-bit proprietary PIC core	32-bit ARM® Cortex®-M0
Pin count	18 to 100	20 to 100
Interrupts	2 ISR with hardware context save	38 ISR with hardware context save, tail-chaining, nesting
Performance	16 MIPS @ 64 MHz	48 MIPS @ 48 MHz
Instructions	83, 16/32 bits	56, 16/32 bits
Program memory	8 o 128 Kbytes	16 to 256 Kbytes
Data memory	256 bytes to 4 Kbytes	4 to 32 Kbytes
Hardware stack	32 levels	No hardware stack but stack as part of RAM memory, thus size configurable
Features	Not available	32x32 hardware multiplier DMA Crystal less USB Analog comparators
	Ethernet	Not available
Highlights	High performance, optimized for C programming, advanced peripherals	Entry-level 32 bits MCU
Total number of devices	193	68

1.1.1 PIC18F series

This standard PIC18 family runs up to 40 MHz and some devices are 5 V compliant. The PIC18F series offer a high-endurance Flash memory (100 Kcycles) and an embedded EEPROM, targeting applications with premium features.

1.1.2 PIC18J series

The PIC18J series feature slightly higher speed (up to 64 MHz) but the Flash endurance does not go over 10 Kcycles. These devices target cost-effective applications with a high memory size.

1.1.3 PIC18K series

The PCI18K series execute code at 64 MHz speed and reach 16 MIPS, but limited to smaller memory sizes. The applications targeted are also cost-effective.

1.2 How to start the migration

- Identify the closest STM32F0 Series to the PIC18 part number, using the device selector available on the web site (search for X-CUBE-PICTOF0).The objective is to align the hardware package and the application GPIOs requirement.
- Follow step by step the guidelines provided in the different migration parts of this application note. At any point of the migration process, refer to the *www.st.com* website to benefit from detailed documents like:
  - the application notes (AN4080, AN4325) to quickly create the prototype of a new application
  - the STM32Cube library to program the application easily
  - the reference manuals for details on advanced features of the microcontrollers

1.3 Key features of the STM32F0 Series

- The STM32F0 Series and PIC18 family devices are available in similar packages and pin count. The hardware impact of the migration is limited.
- In addition to the 32 bits, the core available in the STM32F0 Series offers more development possibilities, some enhanced peripheral features and an extended memory size. Migrating to this STM32F0 series brings also the option to later extend the application to higher-performance products within this microcontrollers family.

2 Core architectures

2.1 Processor overview

The ARM® Cortex®-M0 processor is the smallest ARM processor available. The very small silicon area, low power consumption and minimal code footprint of the processor enable developers to achieve a 32-bit performance at an 8-bit price point, bypassing the step of 16-bit devices. The entire Cortex®-M0 instructions are quickly masterable and its architecture is C-friendly. The development is then simple and fast. Thanks to the fully deterministic instructions and the interrupt timing of the Cortex®-M0, the response times are easily calculated.

The Microchip PIC18 family is also a C compiler optimized architecture. It is an 8-bit CPU featuring 16-bit instructions and four 32-bit instructions. The PIC18 family offers 75 instructions. Some of those devices include few extended instructions for dedicated software features.

On Cortex® architectures, the majority of the instructions are handled in one or two cycles. On PIC products, it is important to distinguish Tcy (instruction cycle time) and Tosc (internal oscillator period). The execution of the instruction needs 1 Tcy (where 1 Tcy = 4 Tosc).

Memory footprint

The selected core determines the type of instructions used and their length. For a given program, the impact on the memory footprint is major as described in *Table 3*.

Table 3. ARM Example: comparing 16-bit multiply operations across processor architectures

8-bit example competitor A		16-bit example	ARM Cortex-M

mulwf LUK_H
movff PRODH,EXT_H
movff PRODL,EXT_L
movf WRD_L,w
mulwf LUK_L
movff PRODH,C3
movff PRODL,WRD_L
mulwf LUK_H
movf PRODL,w
addwf C3,f
movf PRODH,w
addwfc EXT_L,f
	clrf WREG
	addwfc EXT_H,f
	movf WRD_H,w
	mulwf LUK_L
	movf PRODL,w
	addwf C3,w
	movwf WRD_H
	movf PRODH,w
	addwfc EXT_L,f
	clrf WREG
	addwfc EXT_H,f
		MOV R4,&0130h
		MOV R5,&0138h
		MOV SumLo,R6
		MOV SumHi,R7
		(Operands are moved to and from a memory mapped hardware multiply unit)
			MULS r0,r1,r0

In addition to this particular case, Dhrystone and CoreMark® benchmarks were used to compare the general effects of this migration on a standardized C code. The Dhrystone benchmark requires the initialization of important matrices that do not fit in the PIC18 RAM. The CoreMark® code, well implanted in the industry and less memory demanding, is considered as the best option.

The results showing the differences in code size are summarized in *Table 4*.

Table 4. Memory footprint of the CoreMark code

Memory area	STM32F051xx (bytes)		PIC18F46J50 (bytes)	
	High compiler optimization	No optimization	High compiler optimization	No optimization
Code size	5139	6134	8005(1)	20010
Data size	2126	2140	2485	

1. This value could not be verified. It is the announced improvement obtained by using the professional version of XC8 compiler with maximum optimization

The free of charge tools used for this compilation are respectively the MPLAB® X IDE with XC8 compiler, and the Keil™ IDE with ARM C/C++ version 5 compiler, which is without limitation for any of the STM32F0 Series devices.

Concerning the performance, the official website of the EEMBC (Embedded microprocessor benchmark consortium) explains how an important computing bandwidth is won using Cortex®-M0 core, as described in *Table 5*.

Table 5. CoreMark codes comparison between the PIC18 family and the STM32F0 Series

Memory area	STM32F051xx (48 MHz)	PIC18F46K22 (64 MHz)
CoreMark	105.61	7.23
CoreMark / MHz	2.20	0.11

2.2 Core comparison

Both PIC18 and ARM® Cortex®-M0 processors are based on the Von Neumann architecture, having separated memory area for code and data accessed through a unique bus, but the execution unit is different on those devices. The PIC18 family offers only one working register, usually used as one of the ALU operands and often selected to store the results of the operation. RAM is widely used to store processing data in comparison to the 12x32-bit register present in Cortex®-M0 core. In the PIC18 family, as long as the data RAM is contained in the actual active bank, the access is performed in one CPU cycle and accesses to the others banks require the configuration of the Bank Select Register.

Some other special registers are then available for the 8x8 multiplier (PRODH, PRODL), the status register or the stack pointers. This architecture accesses are more complicated to handle at assembly level than the Cortex®-M0 and its low instruction set, which is another asset for developing or debugging assembly code. Nevertheless, C language is also perfectly supported under Cortex®-M0 core. Different tools exist for this architecture and the competition drives IDE and compiler developers to reach high-quality products.

2.3 A secure, OS friendly and easy to debug architecture

The Cortex®-M0 special registers include the Application Program Status Register (APSR), the Execution PSR (EPSR) and the Interrupt PSR (IPSR). The APSR includes, as in the PIC18, the ALU flags mainly used in the conditional branches.

The IPSR always indicates which interrupt handler is currently under execution, allowing easy debug and differentiation if the same handler is accessed from two interrupts. Based on this IPSR value, the control register knows if the device is currently executing under thread mode or handler mode, giving the possibility also to swap from the Main Stack Pointer to the Process Stack Pointer. This feature is fully functional when running an Operating System, as it gives the possibility to separate the process stack from the interrupts and the kernel stack for example, and makes the target application safer. These privileges and execution state do not have an equivalent on the PIC18 architecture.

2.4 Stack organization

The PIC18 family offers a stack of 31 registers of 21 bits each (size of the program counter). These registers are holding return addresses of each subroutine/interrupt. However, as no other register than program counter is systematically stacked, the working register and status register must be stored in the memory to allow a proper execution recovery.

In the Cortex®-M0, 5 x 32 bits working registers are stacked during context saving (R0-R3, R12) in addition to the program counter, the link register and the status register, allowing an easy and effective subroutines or interrupt calls and no increased code size. Furthermore, the context saving of all those registers is performed in only 16 CPU cycles.

On PIC18, this context saving lasts from 12 to 20 oscillator cycles depending on the executed instruction and when the interruption is triggered.

The fact that the stack is hardware defined may be an issue some applications. The Cortex®-M0 stores the stack in the RAM and the user is free to allocate the desired memory, which offers more flexibility.

2.5 Interrupts

The Cortex®-M0 features 32 interrupt lines versus 2 for PIC18, with hardware or software priority (2 levels for PIC18, 4 levels for Cortex®-M0). The Cortex®-M0 offers the nested interrupt (*Figure 2*) and the tail-chaining features (*Figure 3*) allowing to omit context restore.

If during the execution of an interrupt, a second interrupt with the same or lower priority level is received, the program does not return to the main execution but jumps directly to the next interrupt, saving time by not recovering again the context saving of the main execution.

Figure 2. Nested interrupt handling (example)

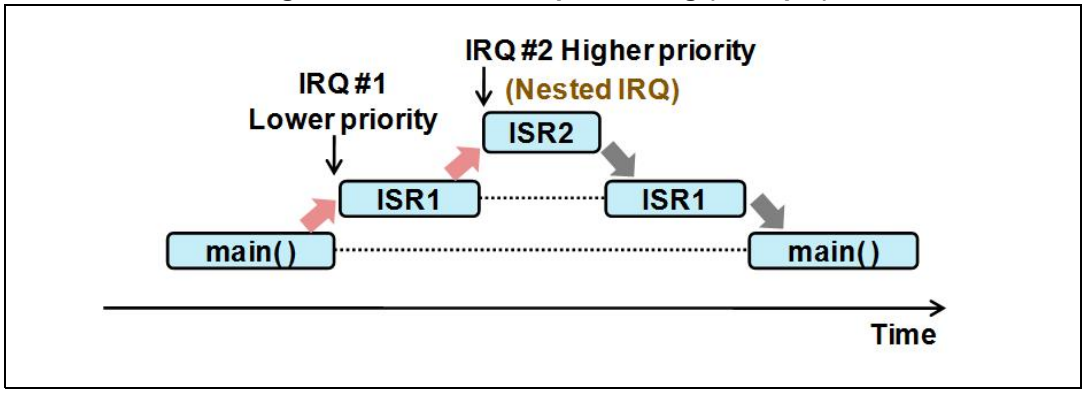
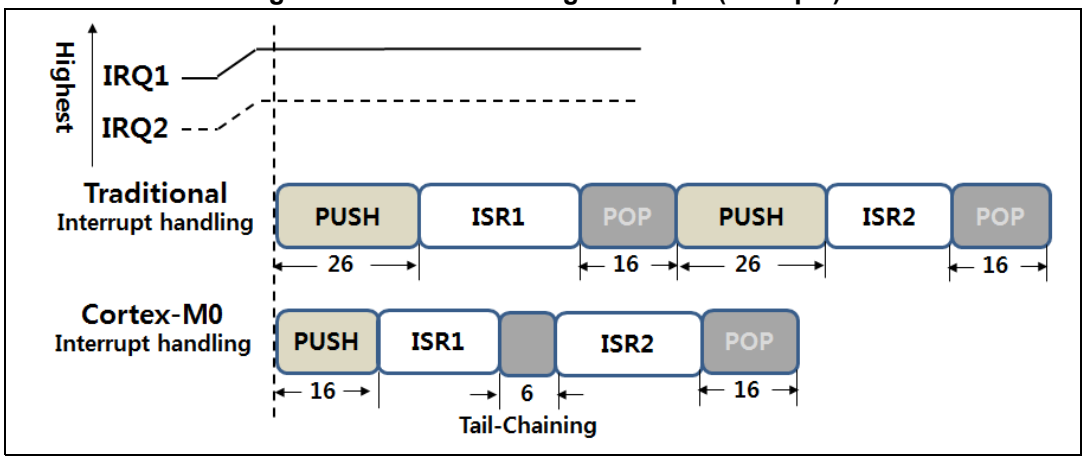


Figure 3. Two tail-chaining interrupts (example)



2.5.1 Internal interrupts

The internal interrupt is quite similar for both PIC18 and Cortex®-M0 architectures if not considering the upper hardware implementation. Thanks to the hardware context saving, the structure below is removed on a Cortex®-M0 device:

```
MOVWF W_TEMP          ; Copy W to a Temporary Register
                      ; regardless of current bank
SWAPF STATUS,W        ; Swap STATUS nibbles and place into W register
MOVWF STATUS_TEMP     ; Save STATUS to a Temporary register in Bank0

: Interrupt Service Routine (ISR)

SWAPF STATUS_TEMP,W   ; Swap original STATUS register value
                      ; into W (restores original bank)
MOVWF STATUS          ; Restore STATUS register from W register
SWAPF W_TEMP,F        ; Swap W_Temp nibbles and return value to W_Temp
SWAPF W_TEMP,W        ; Swap W_Temp to W to restore original
                      ; W value without affecting STATUS
```

2.5.2 External interrupts

The PIC18 family offers three external interrupts to the user, mapped on three pins only. They are edge sensitive (rising or falling) and wake-up the processor from Idle or Sleep modes. INT0 is always higher priority than INT1 & INT2.

The STM32F0 Series offers an external interrupt controller handling up to 31 external event lines. These lines are grouped at the CPU level and the software separates them by checking active value in the EXTI (extended interrupt/event controller) register. The lower 16 event lines are driven independently to any I/O port by the following path: line 1 can be mapped to pin PA1, PB1 … PF1 or PG1, line 2 to PA2, PB2 etc…

2.6 Migration

After having selected the appropriate device in the STM32F0 Series, the user has to start the firmware migration. As described earlier, there is no limitation using the Cortex®-M0 core since moving to 32 bits architecture provides enough bandwidth performance to run any PIC18 code without loss of performance. However, some milestones should be checked to guarantee that the application is able to benefit from the maximum quality offered by the Cortex architecture.

Considering that the code is written in PIC18 assembly language, the user may wish to move to C language in order to increase maintenance productivity and quality. The use of C language allows fast and easy updates plus firmware enhancement. The C language also enables to benefit from legacy codes and worldwide community. However, the code in C language developed or ported by the development team is more readable and understandable than a C language resulting from a decompilation.

Refer to the Boomerang web site for support on how to perform decompilation.

- The integer type is longer on the Cortex architecture. The user has to check how variables are used in the application.
- The interrupt declarations need to be reassigned. Almost all peripherals have a specific ISR. It is no more needed to poll flags in order to catch ISR source which was consuming precious time.

2.7 Key features of the STM32F0 Series

The main advantages of moving to the STM32F0 Series are:

- The performances and the capabilities offered by the Cortex Core
- The reliability of a well implanted and fully trustable platform
- A high number of compilers available, IDEs and an important community

3 System architectures

3.1 Clock

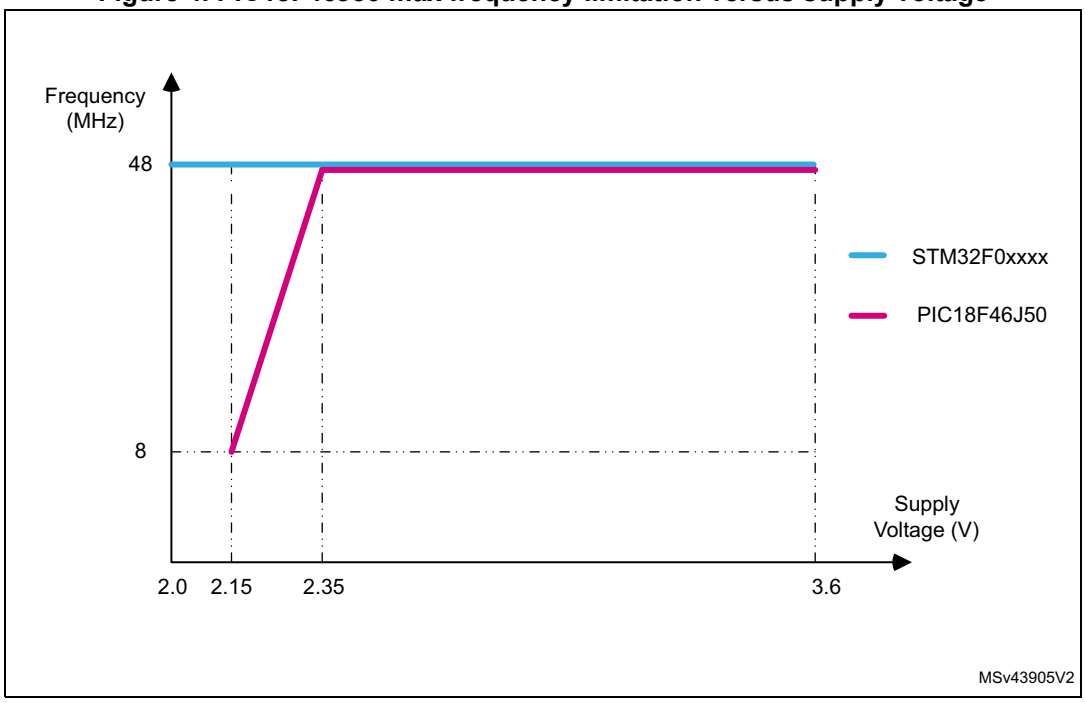
The clock system of the PIC18 Family is controlled mainly through the oscillator module. The PIC18F series embed two internal oscillators: one running at 16 MHz and a low-speed one at 31 kHz.

This setting is similar to the STM32F0 Series entry level although there are two additional oscillators: one at 14 MHz dedicated to the ADC and the other at 48 MHz, on the USB devices, allowing crystal-less functionality.

The PIC18 family PLL is limited to a multiplier by 4, reducing the flexibility and the use cases. Moreover the secondary oscillator input is not always available, reducing the product choice for precise real-time based applications.

The supply voltage is also impacting the allowed frequency, limiting the choice of system clock as shown on *Figure 4*. The frequency of the STM32F0 Series devices is not impacted by supply voltage. The maximum speed is achievable at any supply voltage in the operating conditions.

Figure 4. PIC18F46J50 max frequency limitation versus supply voltage



3.2 Memory

The high-end devices of the PIC18 family are limited to 128 Kbytes of Flash memory and 4 Kbytes of RAM. An additional 1 Kbyte EEPROM is also available but not accessible directly through memory buses. This EEPROM is indirectly addressed through Special Function Registers. The STM32F0 Series competes this feature by using an emulated EEPROM in the Flash memory.

On the performance side, as a CPU cycle is four oscillator cycles on PIC18 family, the Flash memory is accessible without wait state at CPU speed, which means the maximum frequency of the device divided by four. The STM32F0 Series offers a Flash limited to 24 MHz and a prefetch buffer compensating the drawback of wait states inserted over this frequency.

3.3 Power and Reset system

3.3.1 Low-power modes

The seven low-power modes of the PIC18 family are grouped under three major modes, referring to their clock settings.

- The Run mode (similar on the STM32F0 Series)
- The Idle mode (Sleep mode on the STM32F0 Series)
- The Sleep mode (Stop mode on the STM32F0 Series)

The STM32F0 Series offers a Standby mode which has no equivalent in the PIC18 family. When an event is received in low-power mode, the STM32F0 Series wakes up to execute an interrupt or go back to code execution depending on low-power mode instruction (WFI or WFE). Furthermore, the sleep-on-exit feature allows the STM32F0 Series to work in an interrupt mode only. No context saving is performed and all interrupts are tail-chained for a better time efficiency.

3.3.2 Reset

The PIC18 family and STM32F0 Series have similar Reset sources (watchdog reset, power management reset, external reset (MCLR vs NRST) or software reset). Moreover the PIC18 family features a Stack full and Stack underflow reset. The STM32F0 Series stack size is user defined, avoiding overflow leading to a device reset.

Regarding the hardware implementation, the STM32F0 Series contains an internal pull-up resistor on the Reset and it is bidirectional. For example, an internal Watchdog Reset would output a low-level on the NRST pin for a given time lapse.

3.4 Peripherals bus

The 8 bits data bus accesses all peripherals and I/O ports similarly on the PIC18 Von Neumann architecture.

The STM32F0 Bus Matrix and 32 bits bus offer higher bandwidth for any data transfers

3.4.1 GPIOs

The GPIOs of the PIC18 family are 8 bits wide which corresponds to the bus width. They are accessed through the data bus thanks to the Latch register for output, or by direct read when configured in input. The GPIOs implement protection diodes to VDD and VSS that clamp the input voltage. A 5.5V tolerance feature exists, but is not available on the whole family.

The STM32F0 Series offers in addition to previously listed features, more powerful I/O ring, able to deliver higher current or similar with lower voltage drop.

3.4.2 User peripherals

The PIC18 family does not offer the capability to customize the frequency driving the different peripherals. The bus and peripherals connected to it share the same clock. It is different from CPU or gated under sleep mode for example, but it is not possible to have granularity as on the STM32F0 Series to gate it for a given peripheral.

3.5 Migration

This is the main part of the migration process. By definition, core and system architecture are product dependent and thus the code for these parts is more complicated to port from platform to platform. To learn more about how to define a register for a basic usage of a given feature in the STM32F0 Series, refer to the Appendix 1 of the reference manual RM0091. It includes snippets for basic configuration of peripherals through register access.

3.5.1 Clock and Flash setup

The clock tree is easy to migrate, as the PIC18 family and the STM32F0 Series are similar. However, the STM32F0 Series is more customizable in terms of configuration thanks to the flexible PLL and the fact that there is no frequency limitation at low-voltage supply.

For example, the configuration below, interpreted by the compiler to set-up the clock configuration, is replaced by setup of the SetSysClock() function within the SystemInit().

Clock settings for the STM32F0 Series:

```
/* Description: This function enables the interrupt on HSE ready, and start
the HSE as external clock. These settings start and configure the HSE clock
as system clock */
__INLINE void StartHSE(void) {
/* Configure NVIC for RCC */
 NVIC_EnableIRQ(RCC_CRS_IRQn); /* Enable Interrupt on RCC */
 NVIC_SetPriority(RCC_CRS_IRQn,0); /* Set priority for RCC */
/* Enable interrupt on HSE ready */
/* Enable the CSS */
/* Enable the HSE and set HSEBYP to use the external clock instead of an
oscillator */
/* Enable HSE */
/* Note : the clock is switched to HSE in the RCC_CRS_IRQHandler ISR */
 RCC->CIR |= RCC_CIR_HSERDYIE; /* Enable interrupt on HSE ready */
 RCC->CR |= RCC_CR_CSSON | RCC_CR_HSEBYP | RCC_CR_HSEON;/* Enable the CSS */
}
/* Description: This function handles RCC interrupt request and switch the
system clock to HSE. */
void RCC_CRS_IRQHandler(void) {
 /* (1) Check the flag HSE ready */
 /* (2) Clear the flag HSE ready */
 /* (3) Switch the system clock to HSE */
 if ((RCC->CIR & RCC_CIR_HSERDYF) != 0) { /* (1) */
 RCC->CIR |= RCC_CIR_HSERDYC; /* (2) */
```

```
 RCC->CFGR = ((RCC->CFGR & (~RCC_CFGR_SW)) | RCC_CFGR_SW_0); /* (3) */
   }
}
```

Clock settings for the PIC18 MPLAB:

```
// CONFIG1H
#pragma config FOSC = ECHPIO6 // Oscillator Selection bits (EC oscillator (high power, >16 MHz))
#pragma config PLLCFG = OFF // 4X PLL Enable (Oscillator used directly)
#pragma config PRICLKEN = ON // Primary clock enable bit (Primary clock is always enabled)
#pragma config FCMEN = OFF // Fail-Safe Clock Monitor Enable bit (Fail-Safe Clock Monitor disabled)
#pragma config IESO = OFF // Internal/External Oscillator Switchover bit (Oscillator Switchover mode disabled)
```

An important difference between the PIC18 family and the STM32F0 Series is the configuration of the wait state and the prefetch.

Since the PIC18 family requires four clock cycles to perform one read access to the Flash, the memory can be accessed at the system max frequency without problems. On the STM32F0 Series, the CPU executes one instruction per clock cycle, but the Flash frequency is limited to 24 MHz. In order to guarantee a proper execution, a wait state is introduced in the memory access, and a prefetch reduces significantly the drawback of this read delay. The developer can use the two code lines below to configure the prefetch and the wait state to access to the memory at CPU max speed on the STM32F0 Series:

```
/* Enable Prefetch Buffer */
FLASH->ACR |= FLASH_ACR_PRFTBE;
FLASH->ACR |= (uint32_t)FLASH_ACR_LATENCY;
```

3.5.2 GPIO and bus configuration

In the STM32F0 Series, the AHB clock (HCLK) and the APB clock (PCLK) for individual peripherals and memories can be stopped at any time to reduce power consumption. This clock-gating feature requires that, prior to the use of any peripheral, clock is enabled through the RCC_AHBENR, RCC_APB1ENR or RCC_APB2ENR registers.

Then it is possible to configure the GPIOs by setting the MODER register to select the direction or the mode. In order to read input data, use the IDR register. In order to write output data, ODR register has to be used. In case of atomic accesses, BSRR or BRR register is used.

On the PIC18 family, the user has to start preferably to clear the data latch and then configure the TRISC register in input or output mode as desired. Only Port A & B can be used as analog. Only port B allows to set a pull-up, whereas the STM32F0 Series offers this possibility (either pull-up or pull-down) on all GPIOs.

GPIO Initialization comparison between the PIC18 family and the STM32F0 Series

PIC18 source code

```
CLRF PORTC ; Initialize PORTC by clearing output data latches
CLRF LATC ; Alternate method to clear output data latches
MOVLW 0CFh ; Value used to initialize data direction
MOVWF TRISC ; Set RC<3:0> as inputs; RC<5:4> as outputs; RC<7:6> as inputs
```

STM32F0 source code

```
/* (1) Enable the peripheral clock of GPIOA, GPIOB and GPIOC */
/* (2) Select analog mode for PA1 */
/* (3) Select analog mode for PB1 */
/* (4) Select analog mode for PC0 */
RCC->AHBENR |= RCC_AHBENR_GPIOAEN |
RCC_AHBENR_GPIOBEN|RCC_AHBENR_GPIOCEN; /* (1) */
GPIOA->MODER |= GPIO_MODER_MODER1; /* (2) */
GPIOB->MODER |= GPIO_MODER_MODER1; /* (3) */
GPIOC->MODER |= GPIO_MODER_MODER0; /* (4) */
```

To control the output, use preferably the latch register on the PIC18 family to safely perform read-modify-write. To obtain the same latch behavior on the STM32F0 Series, use the BSRR register available for each port.

3.6 Key features of the STM32F0 Series

- High flexibility offered by the clocking system
- Clock gating features, flexible PLL settings, highly configurable clock tree
- No voltage supply limitation
- Bidirectional Reset line

4 Getting started with software and compilation

Microchip offers some useful software examples on their website, developed on C language or in assembly code. The device is claimed to be C Compiler-ready on the datasheet front page, but all configurations and set-ups are still proposed in assembly code. It is an important difference compared with ST approach where all code and examples are easily understandable using C language.

Within the whole STM32F0 Series, the usage of the Hardware Abstraction Layer (HAL) library offers an easy and straightforward porting from one device to another. To keep a lower description level, the snippets or HAL Low Layer library enables to create easily register levels accesses and basic configuration functions.

4.1 Optional additional settings

In the PIC18 family, some hardware features might be configured thanks to the configuration bits, and thus be set after reset without firmware intervention after a start-up. These bits are located in the configuration memory space and should be accessed using table reads and write. They allow to configure watchdog features for example, the Fail Safe Clock Monitor, and additional parameters such as the Two Speed Start-Up. They are configured at the same time than the device is programmed.

Those features are comparable to the configurable STM32F0 option bytes.

4.2 Initialization code from STM32CubeMX code generator

The code generator embedded in the STM32CubeMX is the perfect tool to create the initialization functions for the project. Indeed, a graphic interface guides through the necessary steps in order to obtain a working environment. The user has to integrate his own functions to get desired behavior. If some settings have to be modified or if an other peripheral has to be added, the generator can be accessed afterward. The platform hands-on is extremely simplified.

4.3 Migration

The standard tool used when developing with the PIC18 family is MPLAB. In the trial to port an application described in this document, MPLAB C IDE was used as starting point, and the PIC18F series starter kit targeted. In order to explain how to get started with the basic configuration of the STM32F0 Series and to understand the system behavior, the below paragraphs detail the step-by-step the migration of a simple application based on the most used features in a microcontroller platform (A2D conversion, Input-output, Timer and Interrupt servicing).

Step 1: Getting ready to use workspace

Different IDEs exist for the STM32F0 Series. µVision® from Keil is a fully free of charge tool for these microcontrollers, available on the ARMKeil website. µVision® features its own compiler (downloaded with the IDE) and installs the ST-Link driver that allows to program and debug firmware within the STM32F0 embedded Flash memory.

At this level, the user chooses if he plans to develop the program using a low-level approach based on snippets, or using the STM32CubeMX code generator and the HAL Library. The low-level approach based on snippets allows a full visibility of the hardware and more flexibility. The STM32CubeMX code generator and the HAL library provide a graphical interface for the configuration of the part, resulting in ease of use but higher abstraction of the microcontroller hardware.

Once the toolchain is installed and ready to use, the desired library must be downloaded from the ST website. The easiest approach is to start the project based on one of the multiple projects available in the example folder of the HAL Library. The project must be built for the desired hardware (Nucleo, Discovery, EVAL-Board). If the case snippets / HAL_LowLevel is used, the user needs to integrate them in the project using the same process.

Step 2: Starting configuration

Once the environment is ready, the starting point of the porting is the basic configuration of the device. Watchdog is by default disabled on the STM32F0 Series. In order to have it work at startup by hardware, the user needs to configure the option bytes. µVision® does not enable the modification of the option bytes directly through the programmer. To do so, the user has to download from *www.st.com* the ST proprietary software called STLink-Utility (search for STSW-LINK004).

The system initialization at first level is done through the function SystemInit(). After the reset, it is located in system_stm32f0xx.c module. By default, start is done on the internal RC oscillator @ 8MHz. To speed up the system, switch ON the internal 48 MHz or the PLL and configure them as system clock by using function: SystemClock_Config().

Step 3: System configuration

The system configuration for the PIC18 includes the setup of the main peripherals necessary to properly run this small application. In opposition to the PIC18, where the most classic way is to directly set the register one by one, the HAL Library is used as it offers an easy way to configure each peripheral. First, create and setup a handle for a given instance of the peripheral. In a second time, a structure allows the easy setup of each feature of a given peripheral and starts the whole peripheral afterwards.

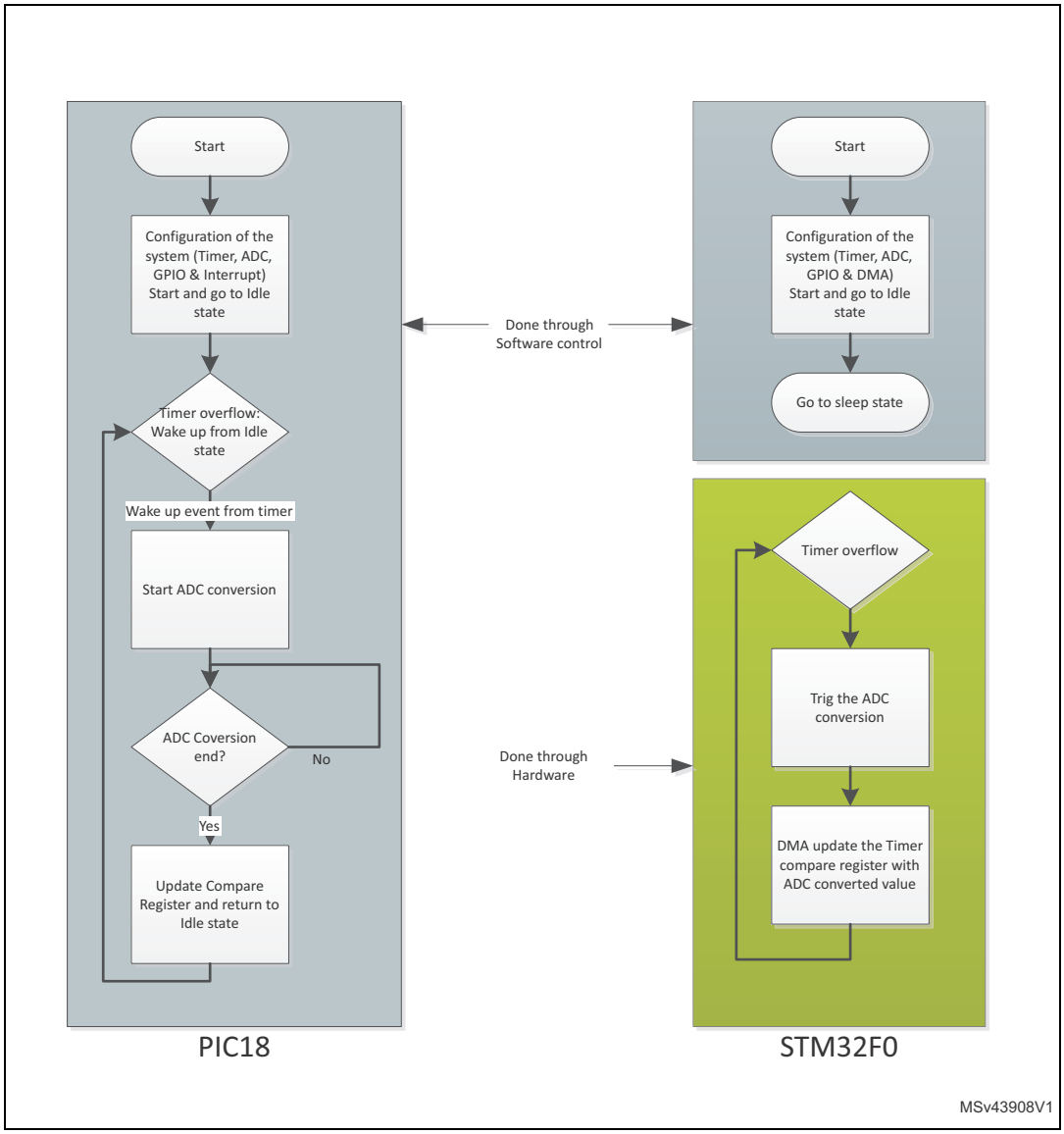
Step4: Execution

A main advantage of the STM32F0 Series compared with the PIC18 family is the capability to interact directly between peripherals without requirement of the core itself. The Timer in PWM mode trigs for example the ADC conversion start. On PIC18 family this feature is available but not usable at the same time as the PWM mode.

Another advantage of the STM32F0 Series is the capability of the DMA to transfer directly the ADC conversion to the Timer compare register. In the PIC18 family, the core must handle this transfer, which implies a loss of computation power when it happens frequently.

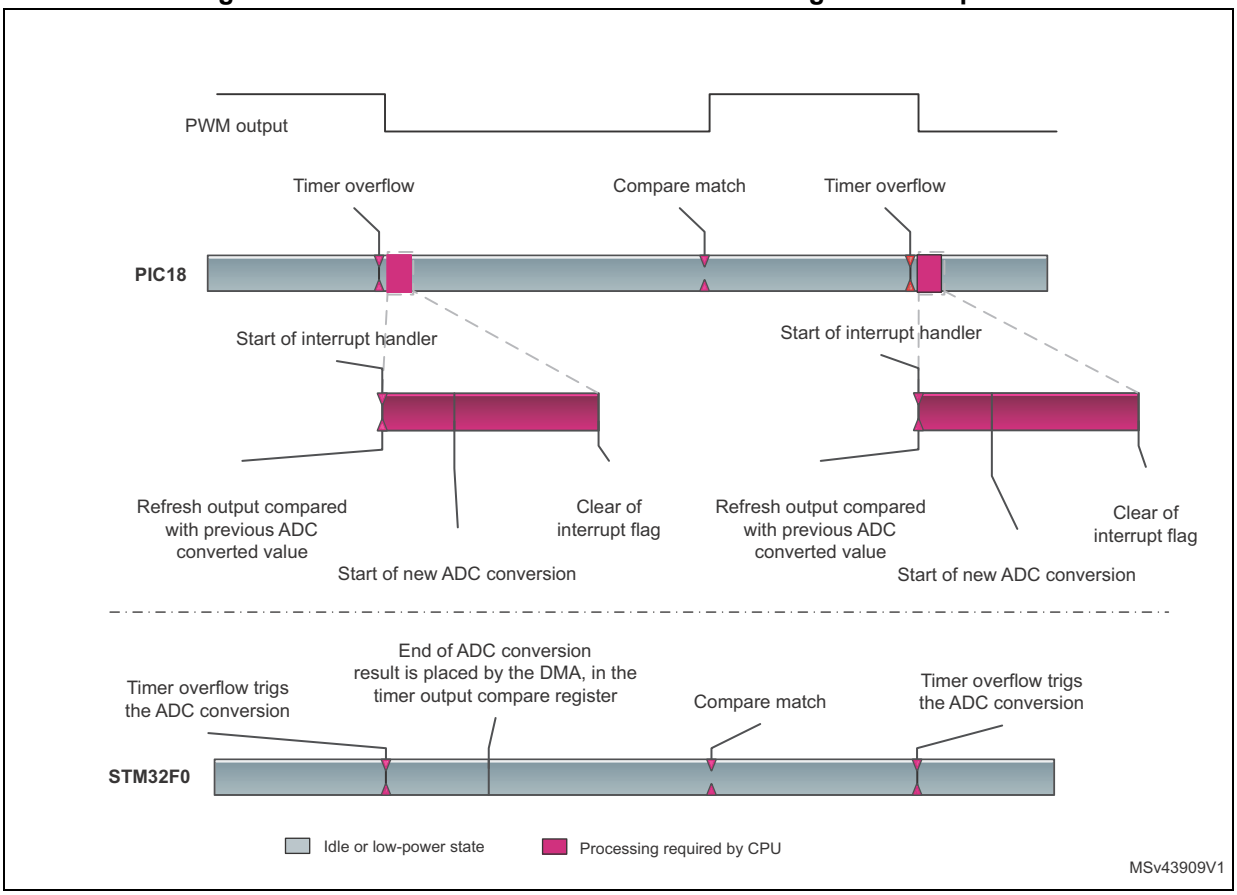
The possible implementation of this dimming code is presented in the flowchart *Figure 5*.

Figure 5. Flowchart of execution of the dimming LED code



The timing diagram showing execution of the software is displayed in *Figure 6*.

Figure 6. Timeline of the behavior of MCUs during one timer period



4.4 Key features of the STM32F0 Series

- A high interconnectivity between the embedded features that the STM32F0 Series offers. It reduces the code memory footprint and the energy of the application.
- The HAL library allows to benefit from the advanced capability of each peripherals of this MCU.

5 Ecosystem

The evolution of the STM32 portfolio provides the opportunity for ST to enlarge its offer concerning the ecosystem.The objective is to ease the handling of the STM32 platforms and to provide to the developers a cross-compatibility between the different devices of the family. Furthermore, the establishment of the Cortex®-M0 core as a main MCU platform, offers the benefit of a large scale of different compilers and IDEs available.

5.1 MPLAB compared to Keil / IAR™ / SW4STM32 compilers

The MPLAB X might be used with the XC8 compiler, in a free version from Microchip website but with drastic limitations, as the full memory size and the speed optimization are available only on the paying version. The STM32F0 Series benefits from a fully free version of Keil IDE with compiler/debugger/programmer, featuring maximum performances.

Developers familiar with the Eclipse-based development environments can benefit from the SW4STM32 IDE and the GCC compiler for ARM cores.

In addition, an extensive set of advanced tools are available for developers seeking particular features.

5.2 MPLAB Code Configurator vs STM32CubeMX

In the migration example described in this document, MPLAB tool has been used, which is more assembly oriented coding and which presents simple functionalities. An advanced tool integrating some more flexibility for the PIC18 is the MPLAB X. This paragrah provides a comparison between the MPLAB X tool (and its integrated Code Configurator) with the STM32CubeMX proprietary tool, which offers a similar modular approach.

At first glance, both tools offer a top view of the selected package. They also offer a menu to select or deselect peripherals and to set up their initial configuration, as shown in *Figure 7* and *Figure 8*.

Figure 7. MPLAB Code Configurator window

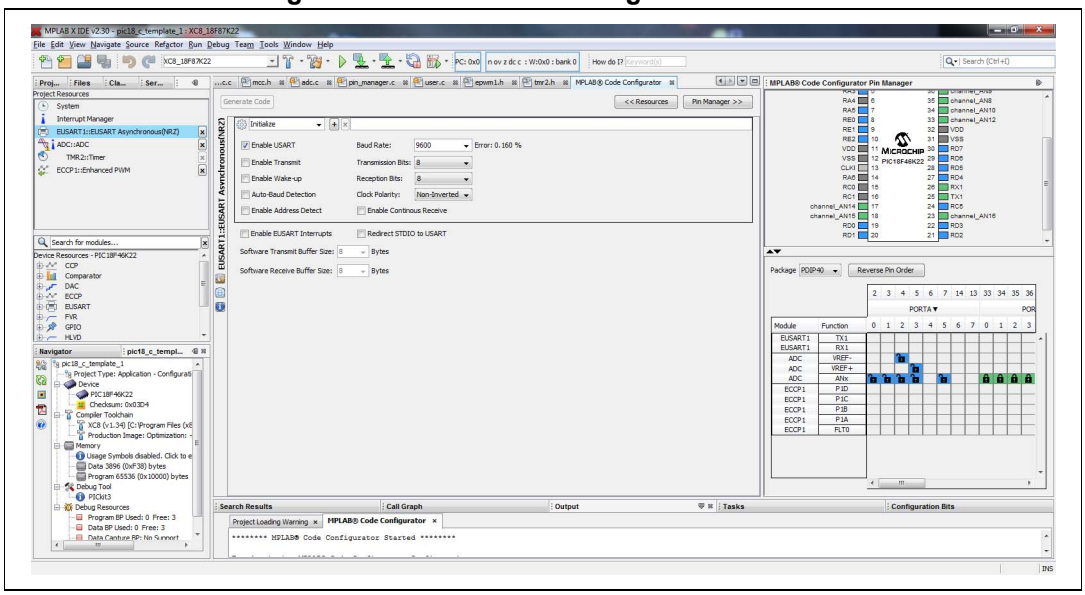
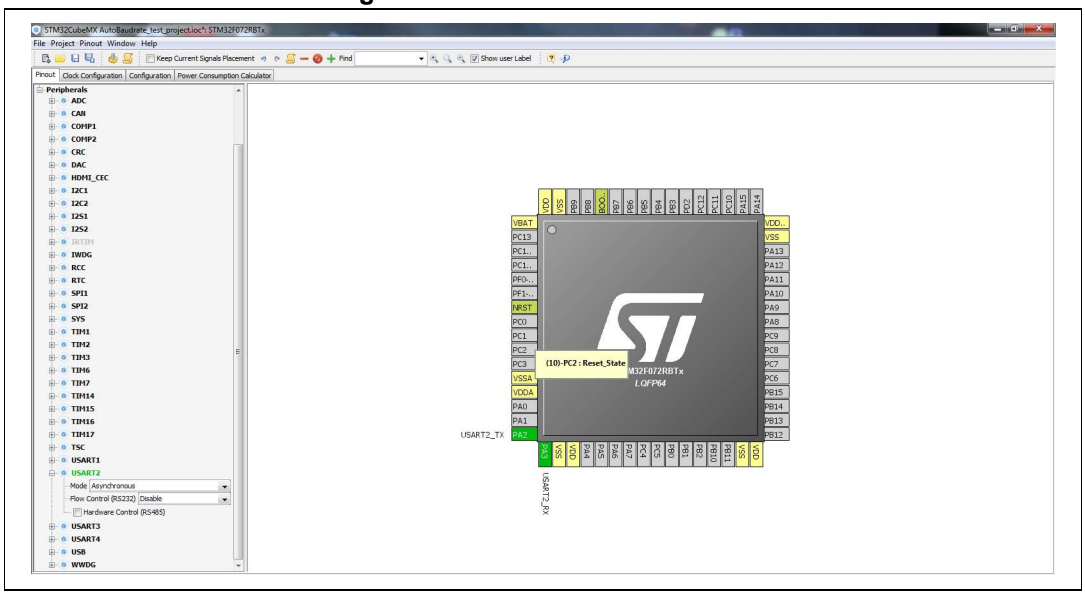


Figure 8. STM32CubeMX window



The Code Configurator tool from Microchip is directly integrated in MPLAB X IDE. The PIC18 as a proprietary core can not be used on different third parties IDE. The STM32CubeMX offers a tool generating projects for most of the compatible IDEs available on the market. This feature accelerates the choice/migration from one IDE to an other as the user does not need to deal with projects porting and reconfiguration of the main options. *Figure 9* shows the initialization option of the serial communication peripheral and the different parameters available.

Figure 9. Example of serial communication setting on STM32CubeMX

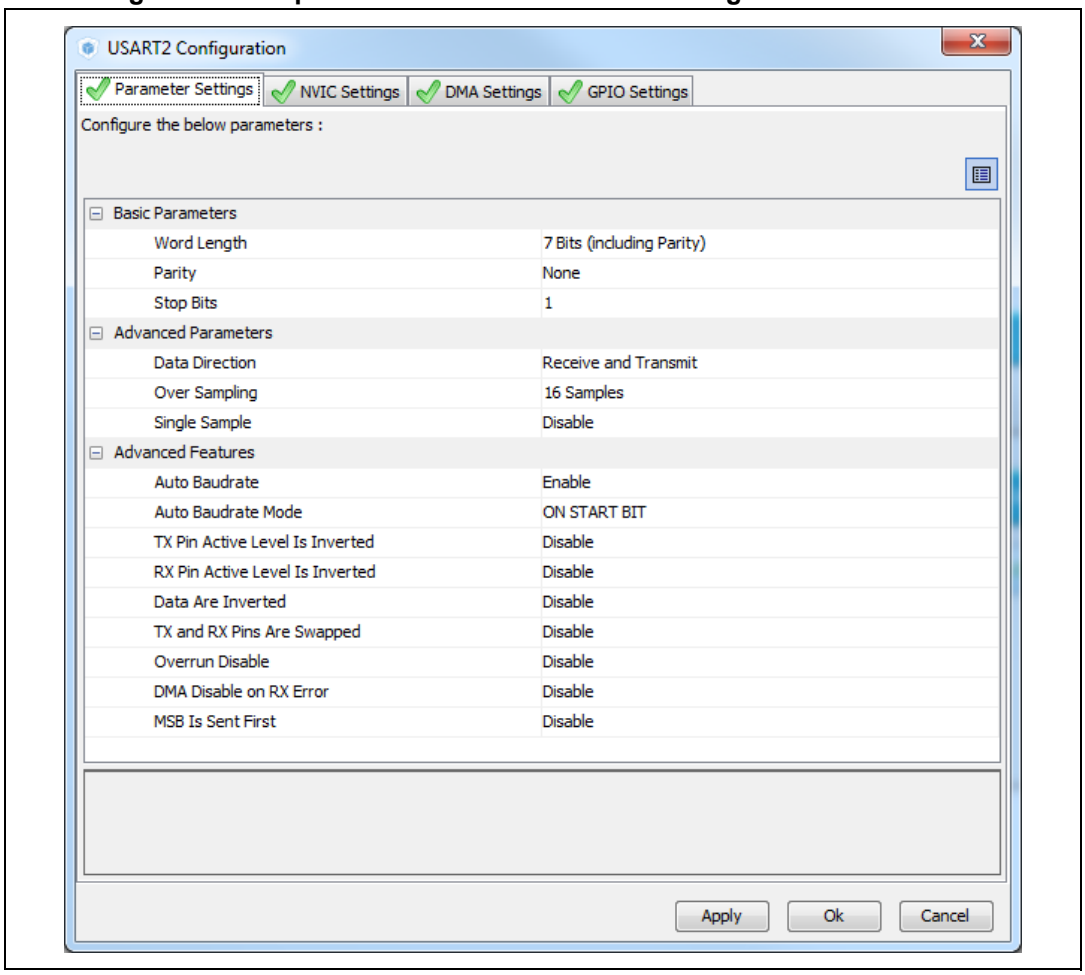
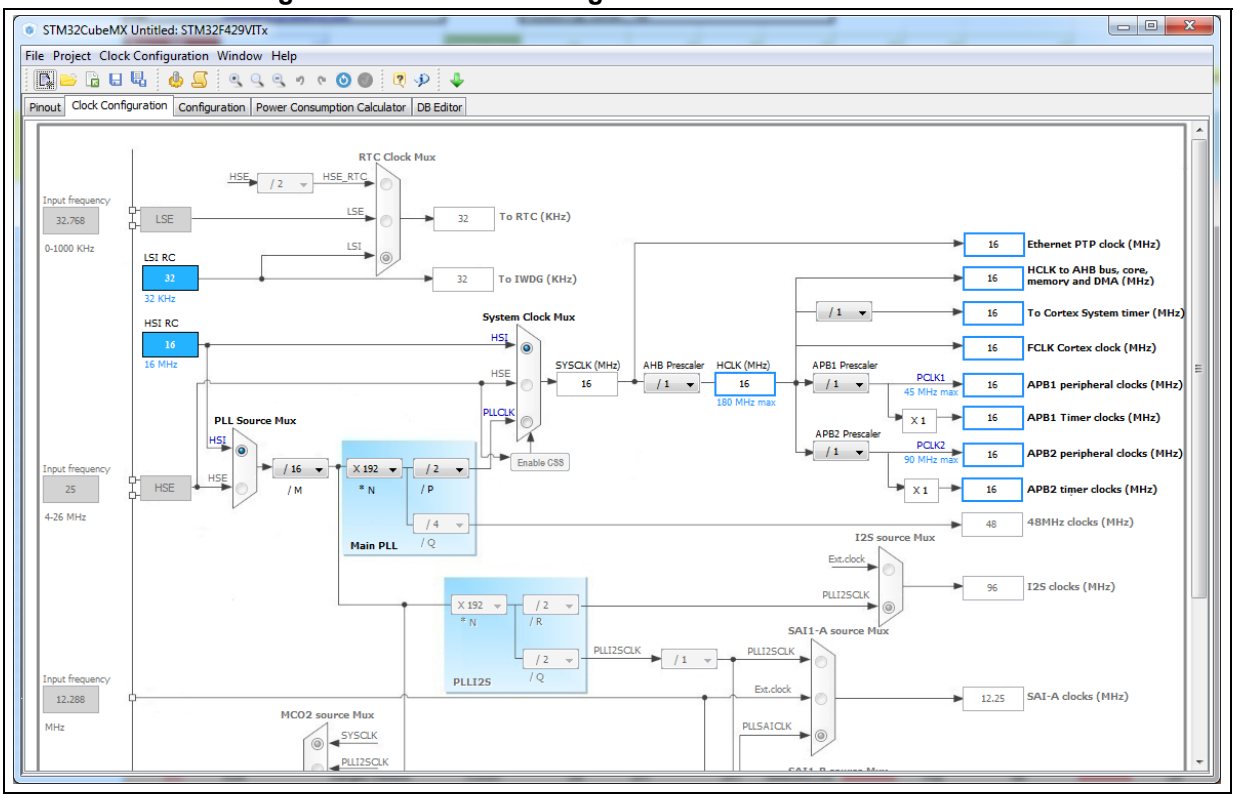


Figure 10. Clock tree configuration tool in STM32CubeMX



The STM32CubeMX integrates a complete clock configuration tool featuring a graphical interface representing the clock tree and its different settings. It simplifies and synthesizes all the possible combinations that can be used to clock each peripheral on the device. It optimizes power consumption, performances and development time.

The Middleware or RTOS can be integrated directly from the configuration or the pinout tabs iinto the library at project generation step. For instance a file system can be added and configured through the tool securing much easier accesses when developing the code.

Another helpful feature available in STM32CubeMX with no comparison in the MPLAB Code Configurator tool is the Power Consumption Calculator. This interface computes, based on the actual system settings and the datasheet values, the current consumption of the system. The developer evaluates the power budget and select good devices for the application.

5.3 Hardware available

A major advantage of the STM32 platform compared with Microchip PIC18 is the consistency of the offer in terms of evaluation boards. For the STM32F0 Series as well as for the other ST MCUs, three different boards type are available:

Nucleo Boards: allow an easy evaluation phase in order to choose the most suitable device, with an extremely simple PCB (2 buttons, 2 LEDs) and an identical pinout between all devices. The Nucleo boards are Arduino compatible and feature many extension boards to obtain the desired option for the system (Motor control, Bluetooth, Sensors, etc…)

Discovery Kits: implement the hardware components to highlight and test the key feature of each device and an extensive pad access to allow the users to get quickly started with the development.

Eval Boards: integrate all the necessary hardware to analyze deeply each peripheral available on the device and give first solutions to PCB designers on how to implement every hardware feature necessary for their application. Firmware developers use these evaluation boards to easily test and challenge their code in the complete system.

PIC18F Starter Kit: the only hardware available for that family, limiting the flexibility and the prototyping simplicity.

5.4 Key features of the STM32F0 Series

The migration is supported by the two important parameters below:

- The Keil IDE and the ARM compiler available without any memory size constraint for the whole STM32F0 family.
- The Arduino compatibility opens the platform to an important community of developers creating many STM32 applications.

6 Revision history

Table 6. Document revision history

Date	Revision	Changes
21-Nov-2016	1	Initial release.
29-Nov-2016	2	Updated *Introduction on page 1* with *Table 1: Applicable products*