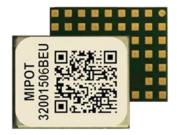


Wirelesxs Dual Core MCU Modules 32001506xxx/32001552xxx

Application note

Dual Core Programming Guide



Description

The 32001506xxx is a multi-protocol sub-GHz radio module based on STM32WL55JC asymmetric dual core arm[®]Cortex[®]-M4/Cortex[®]-M0+ microcontroller. It keeps arm[®]Cortex[®]-M4 core, flash and RAM memory resources and most of the internal peripherals free for the user to integrate his own application without an additional host microcontroller.

Current programming guide targets application developers. It gives an overview of the 32001506xxx module's architecture and provides information about how to create a new project or import an existing one to start developing a user application for the arm[®]Cortex[®]-M4 core. Furthermore it shows how to interact with preloaded RF radio stack fully controlled by arm[®]Cortex[®]-M0+ core from user application residing on arm[®]Cortex[®]-M4 core side.



Contents

1.	Ρ	rerequisites3					
2.	Α	rchitecture overview4					
3.	3. Features						
Э	8.1.	. arm [®] Cortex [®] -M0+ core					
Э	3.2.	. arm [®] Cortex [®] -M4 core					
3	3.3.	. IPCC – Inter-Processor Communication Controller					
4.	С	reating a new STM32 project for arm [®] Cortex [®] -M46					
5.	Ir	mporting an existing STM32 arm [®] Cortex [®] -M4 project13					
6.	D	ebugging STM32 arm [®] Cortex [®] -M4 code15					
	Flash memory and RAM considerations1						
7.	F	lash memory and RAM considerations18					
7. 8.		lash memory and RAM considerations18 TM32 arm®Cortex®-M0+ boot20					
	S	-					
8. 9.	S	TM32 arm [®] Cortex [®] -M0+ boot20 nter-core communication					
8. 9.	S [.] Ir	TM32 arm [®] Cortex [®] -M0+ boot20 nter-core communication					
8. 9.	S Ir).1.	TM32 arm®Cortex®-M0+ boot					
8. 9.	S Ir 9.1. 9.2. 9.3.	TM32 arm [®] Cortex [®] -M0+ boot					
8. 9. 2 2 2 10.	S Ir).1.).2.).3.	TM32 arm®Cortex®-M0+ boot 20 nter-core communication 21 . Communication messages 21 . Service messages 21 . Service messages 22 . Asynchronous messages 24					



1. Prerequisites

Several software development tools may be used to develop applications based on 32001506xxx module.

Current guide will refer to STM32CubeIDE version 1.10.1 or newer which is part of the STM32Cube ecosystem and provides a development platform with peripheral configuration, code generation, code compilation, and debug features for STM32 microcontrollers.

A knowledgebase of STM32WL55JC internal architecture and basic microcontroller programming concepts are required to start developing applications with 32001506xxx module.

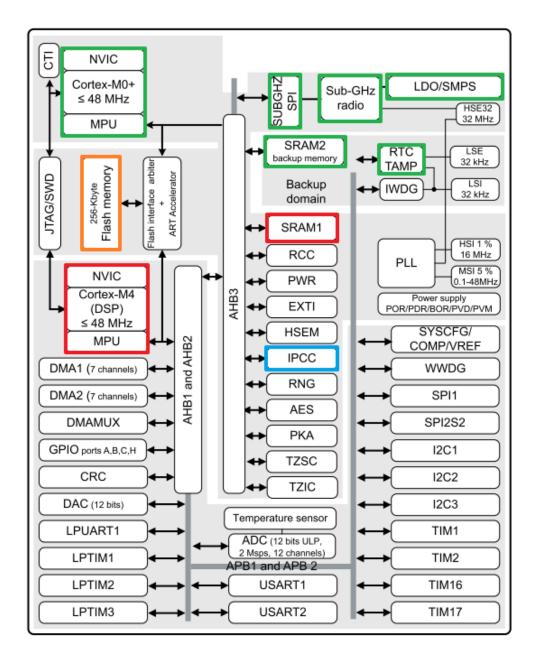
For detailed information about STM32WL55JC microcontroller architecture and peripherals please refer to datasheet "DS13293 Multiprotocol LPWAN dual core 32-bit arm®Cortex®-M4/M0+ LoRa®, (G)FSK, (G)MSK, BPSK, up to 256KB flash, 64KB SRAM" and to "RM0453 Reference manual - STM32WL5x advanced arm®-based 32-bit MCUs with sub-GHz radio solution".

arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere. STM32Cube is an STM trademark.



2. Architecture overview

The following image highlights the STM32WL55JC internal main parts involved in current guide.



32001506xxx module's architecture requires that green bordered parts are exclusively managed by the arm[®]Cortex[®]-M0+ core and can be considered as a "black box" driven by arm[®]Cortex[®]-M4 core (red part) through IPCC (Inter-Processor Communication Controller, blue part).

Of course internal flash memory (orange part) is shared among two cores meaning that part of it is reserved for arm[®] Cortex[®]-M0+ radio application and all remaining is for user application running on arm[®]Cortex[®]-M4. Each core uses its flash memory section in an exclusive mode and is not allowed to operate outside.



3. Features

3.1. arm[®]Cortex[®]-M0+ core

The arm[®]Cortex[®]-M0+ is reserved for sub-GHz radio management and implements the radio stack required by the specific version of the 32001506xxx module. It uses the RTC for stack timing requirements and 32 kB SRAM2 block as volatile working memory; flash memory is shared with arm[®]Cortex[®]-M4 core in the terms already mentioned.

3.2. arm[®]Cortex[®]-M4 core

The arm[®]Cortex[®]-M4 is fully available for user application together with all the peripherals not used by the radio part (all except sub-GHz radio components and RTC). It uses 32 kB SRAM1 block as volatile working memory and shares flash memory with the arm[®]Cortex[®]-M0+ core.

3.3. IPCC – Inter-Processor Communication Controller

IPCC is used to perform bidirectional communication between cores. It is an ST Microcontroller proprietary inter-core communication controller. For details please refer to *"RM0453 Reference manual - STM32WL5x advanced arm®-based 32-bit MCUs with sub-GHz radio solution"*.

IPCC communication operates on a common RAM memory area shared between arm[®]Cortex[®]-M4 and arm[®]Cortex[®]-M0+. In 32001506xxx a 1 kB area is reserved starting from address 0x20008000 to address 0x200083FF. The IPCC shared memory is totally inside SRAM2 block, so it does not affect arm[®]Cortex[®]-M4 available RAM.

IPCC architecture is based on 12 communication channels, 6 of them in the direction from arm[®]Cortex[®]-M4 to arm[®]Cortex[®]-M0+ and remaining 6 in the opposite direction.

32001506xxx module uses 2 pairs of channels in half-duplex mode. The first pair, named *communication channel*, is used for main communication protocol messages, e.g. configuration, transmit or receive commands. The second pair, named *service channel*, is used for internal service messages, e.g. low power management. Each channel operates on its own defined buffer.

The communication over the communication channel between the arm[®]Cortex[®]-M4 and arm[®]Cortex[®]-M0+ cores is performed with the same communication protocol defined for the physical SPI/I²C/UART channels used into 32001505xxx module and is internally managed through transmit/receive interrupts and notifications. For details please refer to *"32001505xxx_Command_Reference_revx.x.pdf"* document and to the following paragraph *"9.1. Communication messages"*.

Protocol messages are written/read to/from IPCC buffers located inside shared RAM area.

A simple arm[®]Cortex[®]-M4 application template may be provided as basic reference to implement IPCC communication with arm[®]Cortex[®]-M0+ core.



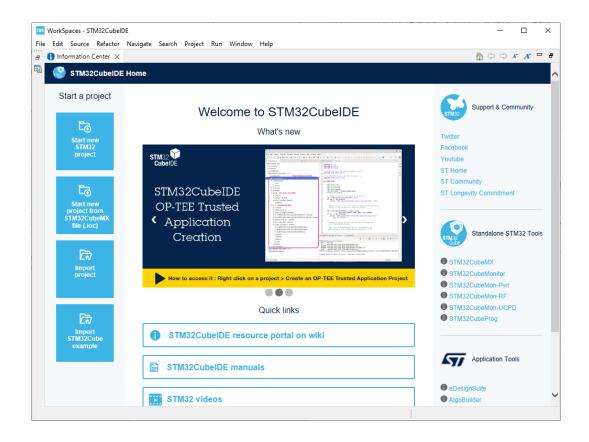
4. Creating a new STM32 project for arm®Cortex®-M4

If not already done, please download and install STM32CubeIDE development platform. It can be found at following link <u>https://www.st.com/en/development-tools/stm32cubeide.html</u> where also user manual and installing guide are available.

After launching STM32CubeIDE select a workspace path. It is the working folder pathname where the project will be created.

IDE STM32Cu	belDE Launcher	\times
Select a dire	ctory as workspace	
STM32Cube	DE uses the workspace directory to store its preferences and development artifacts.	
Workspace:	C:\DATA\WorkSpaces	
Use this a	the default and do not ask again	
Recent Wo	orkspaces	
	Launch Cancel	

To create a new project press *Start new STM32 project* button from the *Information Center* tab:



Then select the part number for which create the new project and press *Next* button:



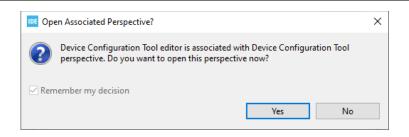
	xample Selector	Cross Selector				
/MPU Filters 📩 🔂 🔂 ン)	Features	Block Diagram Docs	& Resources	Datasheet	📑 Buy
Commercial Part Number STM32WL55JC	~	STM32WL Series				
۹	+-	STM32WL55JCI6		RAM. LoRa, (G)FSK,	Cortex-M4/M0+ @48 MHz with 256 (G)MSK, BPSK modulations. AES 2	
RODUCT INFO	~	ACTIVE	Unit Price for 10kU (US\$) : 5.5143			
Segment	>	Product is in mass production	Boards: NUCLEO-WL55JC1 - NUCLEO-WL	.55JC2	UFBGA 73 5x5x0.6 P 0.5 mm	
Series	>	The STM32WL55/54xx long-range	e wireless and ultra-low-power devices embe	ed a powerful and ultra-low	-power LPWAN-compliant radio solution er	abling the
t la s						
Line	>	following modulations: LoRa [®] , (G				
	>	The LoRa [®] modulation is available	le in STM32WLx5xx only.			
Marketing Status		The LoRa [®] modulation is available These devices are designed to be	le in STM32WLx5xx only. e extremely low-power and are based on the			
Marketing Status Price	>	The LORa [®] modulation is availab These devices are designed to be up to 48 MHz. This core implement independent memory protection u	e in STM32WLx5xx only. e extremely low-power and are based on the tts a full set of DSP instructions. It is compler init (MPU) that enhances the application sec	mented by an Arm [®] Cortex urity.	*-M0+ microcontroller. Both cores impleme	
Marketing Status Price Package	> >	The LORa [®] modulation is availab These devices are designed to be up to 48 MHz. This core implement independent memory protection u	e in STM32WLx5xx only. e extremely low-power and are based on the nts a full set of DSP instructions. It is complet	mented by an Arm [®] Cortex urity.	*-M0+ microcontroller. Both cores impleme	
Marketing Status Price	> > >	The LORa [®] modulation is availab These devices are designed to be up to 48 MHz. This core implement independent memory protection u	e in STM32WLx5xx only. e extremely low-power and are based on the tts a full set of DSP instructions. It is compler init (MPU) that enhances the application sec	mented by an Arm [®] Cortex urity.	*-M0+ microcontroller. Both cores impleme	ent an
Marketing Status Price Package Core	> > > >	The LoRa [®] modulation is availab These devices are designed to b up to 44 MHz This core implement independent memory protection u The devices embed high-speed m MCUs/MPUs List: 2 items	e in STM32WLx5xx only. e extremely low-power and are based on the its a full set of DSP instructions. It is complete init (MPU) that enhances the application sec memories (256-Kbyte Flash memory, 64-Kbyte + Display similar items	mented by an Arm [®] Cortex urity. le SRAM), and an extensiv	[®] -M0+ microcontroller. Both cores impleme e range of enhanced I/Os and peripherals.	ent an
Marketing Status Price Package Core Coprocessor	> > > >	The LoRa® modulation is availab These devices are designed to b up to 44 MHz. This core implement independent memory protection o. The devices embed high-speed MCUs/MPUs List: 2 items	e in STM32WLX5xx only: extremely low-power and are based on the its a full set of DSP instructions. It is complet init (MPU) that enhances the application see memories (256-KDyte Flash memory, 64-Kby + Display similar items Part No Reference MM	mented by an Arm [®] Cortex urity. le SRAM), and an extensive rketing sta× Unit Price f×	®-M0+ microcontroller. Both cores impleme e range of enhanced I/Os and peripherals. Board × Package × Plash	ent an
Marketing Status Price Package Core Coprocessor JEMORY	> > > > >	The LoRa [®] modulation is availab These devices are designed to b up to 44 MHz This core implement independent memory protection u The devices embed high-speed m MCUs/MPUs List: 2 items	e in STM32WLx5xx only. e extremely low-power and are based on the its a full set of DSP instructions. It is complete init (MPU) that enhances the application sec memories (256-Kbyte Flash memory, 64-Kbyte + Display similar items	mented by an Arm [®] Cortex unity. te SRAM), and an extensive rketing Sta× Unit Price f× tive 5.5143	[®] -M0+ microcontroller. Both cores impleme e range of enhanced I/Os and peripherals.	ent an
Marketing Status Price Package Core Coprocessor EMORY Tesh = 256 (kBytes)	> > > > >	The LoRa [®] modulation is availab These devices are designed to bu up to 44 MHz. This core implement independent memory protection u The devices embed high-speed m The devices embed high-speed m MCUs/MPUs List: 2 items Commercial Part No STM32WL55JCl6	e in STM32WLX5xx only: e extremely low-power and are based on the its a full set of DSP instructions. It is complete init (MPU) that enhances the application sec- nemories (256-Kbyte Flash memory, 64-Kby + Display similaritems Part No Reference Mit STM32001.65 Ac	mented by an Arm [®] Cortex unity. te SRAM), and an extensive rketing Sta× Unit Price f× tive 5.5143	[®] -MO+ microcontroller. Both cores impleme e range of enhanced I/Os and peripherais. Bourd X Paskage X Flashi Incore Microson UFBGA 73 5x 256 kByter	ent an
Marketing Status Price Package Core Coprocessor EMORY Flash = 256 (kBytes) 256	> > > > >	The LoRa [®] modulation is availab These devices are designed to bu up to 44 MHz. This core implement independent memory protection u The devices embed high-speed m The devices embed high-speed m MCUs/MPUs List: 2 items Commercial Part No STM32WL55JCl6	e in STM32WLX5xx only: e extremely low-power and are based on the its a full set of DSP instructions. It is complete init (MPU) that enhances the application sec- nemories (256-Kbyte Flash memory, 64-Kby + Display similaritems Part No Reference Mit STM32001.65 Ac	mented by an Arm [®] Cortex unity. te SRAM), and an extensive rketing Sta× Unit Price f× tive 5.5143	[®] -MO+ microcontroller. Both cores impleme e range of enhanced I/Os and peripherais. Bourd X Paskage X Flashi Incore Microson UFBGA 73 5x 256 kByter	ent an
Marketing Status Price Package Core Coprocessor EMORY Itash = 256 (kBytes) 256 EEPROM = 0 (Bytes)	> > > > >	The LoRa [®] modulation is availab These devices are designed to bu up to 44 MHz. This core implement independent memory protection u The devices embed high-speed m The devices embed high-speed m MCUs/MPUs List: 2 items Commercial Part No STM32WL55JCl6	e in STM32WLX5xx only: e extremely low-power and are based on the its a full set of DSP instructions. It is complete init (MPU) that enhances the application sec- nemories (256-Kbyte Flash memory, 64-Kby + Display similaritems Part No Reference Mit STM32001.65 Ac	mented by an Arm [®] Cortex unity. te SRAM), and an extensive rketing Sta× Unit Price f× tive 5.5143	[®] -MO+ microcontroller. Both cores impleme e range of enhanced I/Os and peripherais. Bourd X Paskage X Flashi Incore Microson UFBGA 73 5x 256 kByter	ent an
Marketing Status Price Package Core Coprocessor EMORY Flash = 256 (kBytes) 256 ERROM = 0 (Bytes) 0	> > > > >	The LoRa [®] modulation is availab These devices are designed to bu up to 44 MHz. This core implement independent memory protection u The devices embed high-speed m The devices embed high-speed m MCUs/MPUs List: 2 items Commercial Part No STM32WL55JCl6	e in STM32WLX5xx only: e extremely low-power and are based on the its a full set of DSP instructions. It is complete init (MPU) that enhances the application sec- nemories (256-Kbyte Flash memory, 64-Kby + Display similaritems Part No Reference Mit STM32001.65 Ac	mented by an Arm [®] Cortex unity. te SRAM), and an extensive rketing Sta× Unit Price f× tive 5.5143	[®] -MO+ microcontroller. Both cores impleme e range of enhanced I/Os and peripherais. Bourd X Paskage X Flashi Incore Microson UFBGA 73 5x 256 kByter	ent an
Marketing Status Price Package Core Coprocessor EMORY Flash = 256 (kBytes) 256 EPROM = 0 (Bytes) 0 RAM Total = 64 (kBytes)	> > > > >	The LoRa [®] modulation is availab These devices are designed to bu up to 44 MHz. This core implement independent memory protection u The devices embed high-speed m The devices embed high-speed m MCUs/MPUs List: 2 items Commercial Part No STM32WL55JCl6	e in STM32WLX5xx only: e extremely low-power and are based on the its a full set of DSP instructions. It is complete init (MPU) that enhances the application sec- nemories (256-Kbyte Flash memory, 64-Kby + Display similaritems Part No Reference Mit STM32001.65 Ac	mented by an Arm [®] Cortex unity. te SRAM), and an extensive rketing Sta× Unit Price f× tive 5.5143	[®] -MO+ microcontroller. Both cores impleme e range of enhanced I/Os and peripherais. Bourd X Paskage X Flashi Incore Microson UFBGA 73 5x 256 kByter	ent an
Marketing Status Price Package Core Coprocessor IEMORY Flash = 256 (kBytes) 256 EEPROM = 0 (Bytes) 0 RAM Total = 64 (kBytes)	> > > > >	The LoRa [®] modulation is availab These devices are designed to bu up to 44 MHz. This core implement independent memory protection u The devices embed high-speed m The devices embed high-speed m MCUs/MPUs List: 2 items Commercial Part No STM32WL55JCl6	e in STM32WLX5xx only: e extremely low-power and are based on the its a full set of DSP instructions. It is complete init (MPU) that enhances the application sec- nemories (256-Kbyte Flash memory, 64-Kby + Display similaritems Part No Reference Mit STM32001.65 Ac	mented by an Arm [®] Cortex unity. te SRAM), and an extensive rketing Sta× Unit Price f× tive 5.5143	[®] -MO+ microcontroller. Both cores impleme e range of enhanced I/Os and peripherais. Bourd X Paskage X Flashi Incore Microson UFBGA 73 5x 256 kByter	ent an
Marketing Status Price Price Price Core Coprocessor EMORY EMORY ELEMORY 256 EEPROM = 0 (Bytes) 0 RM Total = 64 (kBytes) 64 RAM = 64 (kBytes)	> > > > >	The LoRa [®] modulation is availab These devices are designed to bu up to 44 MHz. This core implement independent memory protection u The devices embed high-speed m The devices embed high-speed m MCUs/MPUs List: 2 items Commercial Part No STM32WL55JCl6	e in STM32WLX5xx only: e extremely low-power and are based on the its a full set of DSP instructions. It is complete init (MPU) that enhances the application sec- nemories (256-Kbyte Flash memory, 64-Kby + Display similaritems Part No Reference Mit STM32001.65 Ac	mented by an Arm [®] Cortex unity. te SRAM), and an extensive rketing Sta× Unit Price f× tive 5.5143	[®] -MO+ microcontroller. Both cores impleme e range of enhanced I/Os and peripherais. Bourd X Paskage X Flashi Incore Microson UFBGA 73 5x 256 kByter	ent an
Price Package Core	> > > > >	The LoRa [®] modulation is availab These devices are designed to bu up to 44 MHz. This core implement independent memory protection u The devices embed high-speed m The devices embed high-speed m MCUs/MPUs List: 2 items Commercial Part No STM32WL55JCl6	e in STM32WLX5xx only: e extremely low-power and are based on the its a full set of DSP instructions. It is complete init (MPU) that enhances the application sec- nemories (256-Kbyte Flash memory, 64-Kby + Display similaritems Part No Reference Mit STM32001.65 Ac	mented by an Arm [®] Cortex unity. te SRAM), and an extensive rketing Sta× Unit Price f× tive 5.5143	[®] -MO+ microcontroller. Both cores impleme e range of enhanced I/Os and peripherais. Bourd X Paskage X Flashi Incore Microson UFBGA 73 5x 256 kByter	ent an

Enter a name for the new project, let the *"Enable Multi Cpus Configuration"* flag selected and then press *Finish* button:

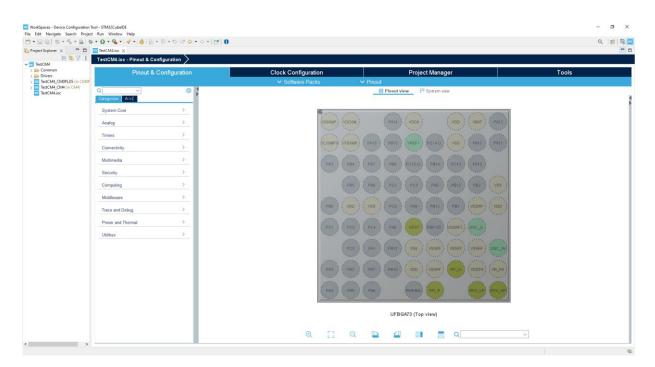
DE STM32 Proje	ct	_	
Setup STM32 pr	oject		IDE
Project			
Project Name:	TestCM4		
Use default	location		
Location:	C:/DATA/WorkSpaces		Browse
Options Targeted La	nguage		
●c ⊖c	**		
Targeted Do	evice Usage Multi Cpus Configuration		
Targeted Bi Executal	nary Type ole O Static Library		
Targeted Pr STM32C	oject Type ube ○Empty		
?	< Back Next >	Finish	Cancel

Press Yes button to the request "Open Associated Perspective?":





The project is then created and the STM32CubeIDE opens the Device Configuration Tool to allow device hardware configuration:



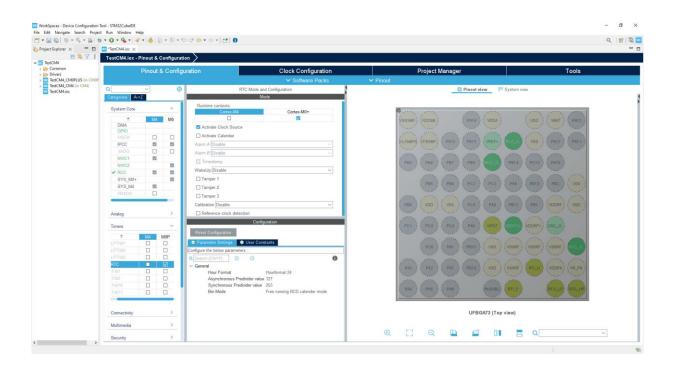
Here it is possible to set pin functionalities, select the peripherals to enable and configure them, configure the clocks for all the peripherals, enable middlewares and so on.

Configuring the clocks is quite trivial because the oscillators are the same for both cores, so no any choices are allowed. The module contains a 32 MHz TCXO as HSE (High Speed External oscillator) main oscillator and a 32.768 kHz crystal used as LSE (Low Speed External oscillator) for the RTC. In the *System Core* category, into *RCC* section select TCXO for HSE and Crystal/Ceramic Resonator for LSE.

In *Timers* category, into *RTC* section enable the RTC for Cortex[®]-M0+ core and set *Activate Clock Source* flag to enable LSE for the RTC.

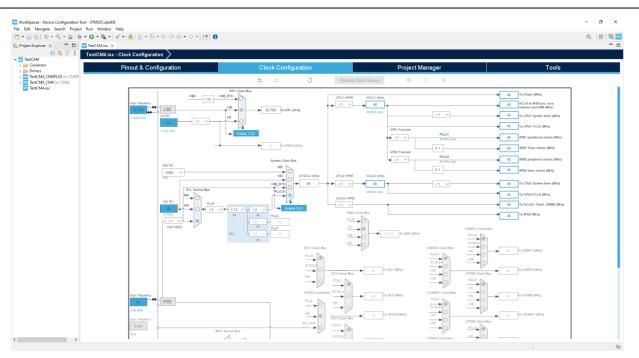


Image: State		TestCM4.ioc ×	· 취 • 약 수 • 수 • 편 8			Q, 😰 🗄							
Conversions Pinout & Configuration Clock Configuration Projet Manager Tools With Conversions Image: Conversions<													
March Markel (Markel Markel) System Cov Solution Solu	Common Drivers	Pinout & C	Configuration	Clock Configuration	Project Manager	Tools							
Certagene A.2 Mode Partene certacti Partene certacti Partene Certacti	> IOI TestCM4_CM0PLUS (in CM0P > IOI TestCM4_CM4 (in CM4)		PCC Mode and Confir			dasse 🗰 Suetam siaw							
System Color Contex Add WAA B WAA B WAA B High Speed Clock (RSE) (C)/Lattice B WAC WCC WAC B WAC B WAA B	TestCM4.ioc	· · · · · · · · · · · · · · · · · · ·				a - Ugarain man							
Analog Analog Times Configuration Consolid Examples Administry Examples Administry Examples Administry Examples Computing Examples Computing Image: Solid Configuration Security Image: Solid Configuration Computing Image: Solid Configuration Middeware Protects Bits balance Statusd Protect Bits Examples Middeware Protects Bits Protect Bits Examples Middeware Protects Bits Protect Bits Examples Middeware Protects Bits Protect Bits Examples Middeware Protect Bits Power and Ther. Middeware Power and Ther. Middeware		oyumi do DMA GPD HSEM HSEM HSEM HSC HNCC NNCC NNCC SYS_M0+ SYS_M4 SYS_M4 SYS_M4	Contex.M4	×									
Connecting Image: Connecting Image: Connecting Image: Connecting Mulmoda Image: Connecting Image: Connecting Image: Connecting Security Image: Connecting Image: Connecting Image: Connecting Computing Image: Connecting Image: Connecting Image: Connecting Middeease Image: Connecting Image: Connecting Image: Connecting Middeease Image: Connecting Image: Connecting Image: Connecting Tace and Debag Image: Connecting Image: Connecting Image: Connecting Pewer and The: MS Collaboritor Wale Image: Connecting Image: Connecting		Timers >			- A A A	NRST NOTCO VODES) (OSC_0.)							
Security Computing Computing 0 Computing 0 Variation Cache Excluded Indexee Practice Backed Trace and Debug Data Cache Panet res V05 (r CDV cycles) Version of the Computing V05 (r CDV cycles) Panet and Ther. V05 (r CDV cycles)		Connectivity >	User Constants		PCS PAT PBTT								
Computing → System Parameters Computing → System Parameters Modelexer → Prefetch Buffer Trace and Debug > Data Cache Panet radiu Table Latercy(VS) WS (1 d'DU cycle) VS (1 d'DU cycle) Veref and Ther> VS Collarion Value HS Calibration Value 64				0									
Middeware Perfecto Buffer Disabled Trace and Debug Data Cache Enabled Pewer and Ther> VRC Parameters UFB GAT3 (Top view)		v	System Parameters VDD voltage (V) 3.3 V		PA3 PA3 PA7 PB10	VOD (VSSRP) ERUN (VDDPA) (VRCPA)							
Trace and Debug 2 Flash Latency(WS) 0 WS (1 CPU cycle) Power and Ther_> HS Calibration Value 64 UFBGA73 (Top view)		Middleware >	Prefetch Buffer Disable	d	PA4 PA5 PA8	PH380 RFLP RF0 P RF0 HB							
Power and Ther? HSI Calibration Value 64 UPDGR/J 3 (169 View)			Flash Latency(WS) 0 WS (1		Contraction (1995)	and a set of the set o							
		Power and Ther >	HSI Calibration Value 64		UFB	GA73 (Top view)							



Then, in the Clock Configuration tab select the LSE source clock for the RTC, select the HSE as main oscillator source clock, enable the PLLCLK and set the following values for PLL multipliers/dividers: PLLM = /4, *N=X12, /R=/2. This way a SYSCLK equal to 48 MHz will be obtained. It is necessary to keep the HCLK2 for CPU2 (Cortex[®]-M0+) equal to 48 MHz. Other clocks may also be changed based on user application requirements.





In the *System Core* category, into *IPCC* section set the flag Activated to enable the interprocessor communication controller.

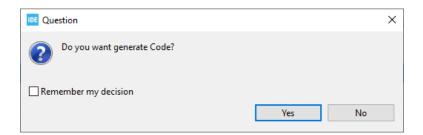
Concerning Conconcerning Concerning Concerning	🗙 🖻 🛱 🎖 🖁 🗖 🔲 🏧 "TestCM4.ioc 🗙 🖻 main.c 🔹 ma													
CMI (= CAR) Image: Table and Configuration Image: Tab		TestCM4.loc - Pinout & Configuration >												
Image: Computing Co		guration	v		Tools									
System Core Control Image: Control Image: Control Image: Contro Image: Con	oc	IPC	C Mode and Configuration		out view									
DAA Image I	System Core		Cortex-M0+	@										
Times > Connectivity > Multimedia > Security > Computing 0 Computing 0 Modeware > Trace and Debug MOZERANCE Table Exable Permettion Protect Sub Priority PCC RX. Occupation 0 Modeware > Trace and Debug >	DMA GP0 MSEM □ MSEA □ MMC1 □ NMC2 □ MSC □ SYS_MS □ SYS_MS □ WWOC □	E Activited	2											
Computing > Middeware > Trace and Debug > NVX2 Interrupt Table Enabled Presentation Proofly PCC RV Occupied and TX Free Interrupt Image: Debug	Connectivity Multimedia	Reset Configuration Parameter Settings NVIC1 Interrupt Table PCC RX Occupied Interrupt	AC Settings Enabled Preemption Priority Sub Pric											
Trace and Debug >> INVC2 Interrupt Table Exabled Preemption Privaty PCC RX Occupied and TX Free Interrupt II 0			0	PA3 PA2 PAT PB	10 VOD (VSSRF) RFLN (VODPA) (VR.PA)									
IPCC RX Occupied and TX Free Interrupt 🔯 0	Middleware	>		PA4 PA5 PAB	PH3-BO, RFLP RFO_LP RFO_HP									
		IPCC RX Occupied and TX Free In			antes antes antes									

In the *Project Manager* tab, into *Code Generator* section the flag *Generate peripheral initialization as pair of '.c/.h' files per peripheral* may be selected to obtain a separation into multiple pairs of files of the low level code generated for each enabled peripheral.



100 WorkSpaces - Device Configuration To					- a ×						
File Edit Navigate Search Project											
		💩 [일 = 종] = 한 각 수 = 아 = [🛃 🚺			Q. 189 🖬 🖬						
	TestCM4.loc - Project Manager										
- 🔁 🍞 🕴	TestCM4.ioc - Proje	ect Manager >									
> 🦢 Common	Pin	t & Configuration Clock Configuration Project Manager Tools									
> 🗁 Drivers > 🚾 TestCM4_CM0PLUS (in CM0P		STM32Cube MCU packages and embedded s	aftware packs								
> III TestCM4_CM4 (in CM4) III TestCM4.ioc		Copy all used libraries into the project folde	r								
iestCM4.loc	Project	 Copy only the necessary library files 									
	Froject	O Add necessary library files as reference in	the toolchain project configuration file								
	Code Generator	Generated files									
		Generate peripheral initialization as a pair of	f .c/,h files per peripheral								
		Backup previously generated files when re-									
		Keep User Code when re-generating									
		Delete previously generated files when not	re-generated								
	Advanced Settings										
		/HAL Settings									
		Set all free pins as analog (to optimize the	power consumption)								
		Enable Full Assert									
		/Template Settings									
		Select a template to generate customized cor	le Settings								
		Select a template to generate customized col	Settings								
< >											

In the *Project Manager* tab, into *Advanced Settings* section, it is possible to choose for each enabled peripheral the library to use for low level code generation. Two choices are available: *HAL* driver library and *LL* driver library which provide a set of low level tools to control peripherals. LL library differs from HAL library in the fact that it provides a complete control of every peripheral's feature allowing the user to directly operate on peripheral's registers. HAL library, instead, is a higher level driver and adds an abstraction layer between hardware and application code which introduces some pre-configured management of the peripherals. When saving the project the device configuration tool asks to generate the low level code:



Answering *Yes*, causes device configuration tool to create a framework project with all enabled peripherals initialized and ready for the user to start developing his own application.



m (m) 🚓 🗸 🤌 🗸 👘 i 🖉 i 🕅 i	• 🚳 • 💽 • 🞯 • 🕸 • 🚺	- 🚱 - 🂁 - 🙆) 🔗 🕶 🍠 🐨 🕤	황이 두 왕이 두 100 리카 -	🗢 🕶 🗢 🖬 📷	0								Q 🛛 😰 🛛 🗔
ct Explorer ×	E 🕏 🍸 🕴 🗖 🖬	TestCM4.ioc	i main.c ×								- 0	🚼 Outline 🗙 💿	Build Targets	
stCM4			CODE BEGIN Header */								^		E	12 R 🕹 • ¥
Common		28 /**										🛀 main,h	0	
Drivers		3 * @fil										ipcc.h		
TestCM4_CM0PLUS (in CM0PLUS)		5 * Shei	of Main ord	eram body								gpio.h		
TestCM4_CM4 (in CM4)		6 ******	*****	*****									ck_Config(void) : void	đ
Includes		7 * @att	ention									 main(void) 	: int	
Section 2018			right (c) 2022 STMicro	alectropics								 SystemClor 	ck_Config(void) : voi:	đ
🧏 Core			rights reserved.									 Error_Handl 		
> 📴 Inc		11 *									- 11	💉 assert_failed	d(uint8_t*, uint32_t):	void
e 🗁 Src			software is licensed he root directory of t			n the LICENSE fi	le							
> in gpio.c > in ipcc.c			ne root directory of t o LICENSE file comes w			ed AS-IS.								
> in inc		15 *												
> (c) main.c > (c) stm32wbx hal msp.c				*****	••••		***							
> a stm32wlxx_it.c		17 */	CODE END Header */											
> is syscalls.c			CODE END Header */ des				-*/							
> (c) sysmem.c		20 #include	"main.h"											
👝 Startup		21 #include												
9 Drivers		22 #include 23	~gp10.h~											
STM32WLxx_HAL_Driver		248 /* Priva	te includes				-*/							
> 🛃 stm32wlxx_hal_cortex.c			CODE BEGIN Includes */											
> 🔒 stm32wlxx_hal_dma_ex.c		26	CODE END Includes */											
> 🔒 stm32wlxx_hal_dma.c		28	CODE END INCIDUES 7											
> 🔒 stm32wlxx_hal_exti.c		290 /* Priva	te typedef				-*/							
> 🙀 stm32wlxx_hal_flash_ex.c			CODE BEGIN PTD */											
> 🗟 stm32wlxx_hal_flash.c		31 32 /* USER (CODE END PTD */											
> A stm32wlxx_hal_gpio.c > A stm32wlxx_hal_ipcc.c		33												
> ktm32wixx_hal_pwr_ex.c		340/* Priva	te define				-*/							
> Re stm32wkx hal pwr.c			CODE BEGIN PD */ CODE END PD */											
> & stm32wixx_hal_rcc_ex.c		30 /* USER 1	CODE END PD -7											
> a stm32wlxx hal rcc.c			te macro				-*/							
> a stm32wlxx_hal.c			CODE BEGIN PM */											
STM32WL55ICIX FLASH.Id		40	CODE END PM */											
TestCM4.ioc		42	CODE END PH 7											
			te variables				-*/							
		44									~			
		🛃 Problems 🗙	🕗 Tasks 🕒 Console 📃	Properties			78 =	🔒 Build Analyzer 🗙	🛓 Static Str	sck Analyzer				28
		0 items												
		Description		Resource	Path	Location	Type	Memory Regions Me	mon Details					
									tart address	End address	Size	Free	Used	11
								Neylon St	are dooress	end address	Size	Free	Used	Usage (%)

Code generation may be requested at any time using the specific icon *ideal*, using the *Alt+K* key shortcut or selecting the *Generate Code* item into *Project menu*.

Proj	ect	Run	Window	Help
	Ор	en Proj	ject	ł
	Clo	ose Proj	ject	s
010	Bui	ild All		Ctrl+B
	Bui	ild Con	figurations	>
	Bui	ild Proj	ect	
	Bui	ild Wor	king Set	>
	Cle	an		
	Bui	ild Auto	omatically	
	Bui	ild Targ	ets	>
	C/(C++ In	dex	>
мх	Ge	nerate l	Report	
٩	Ge	nerate (Code	
	Pro	perties	;	E

The configuration tool generates the code for both arm[®]Cortex[®]-M0+ and arm[®]Cortex[®]-M4 cores. The project for the arm[®]Cortex[®]-M0+ core is needed just to enable the HCLK2 clock for Cpu2 and the IPCC controller but the user does not have to develop any code for it and must not re-program any code into arm[®]Cortex[®]-M0+ reserved flash memory.

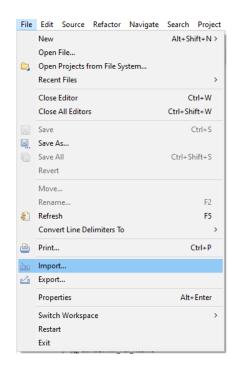


5. Importing an existing STM32 arm®Cortex®-M4 project

To import an existing STM32 arm[®]Cortex[®]-M4 project press *Import project* button from the *Information Center* tab:

WorkSpeces - TestCM4_CM0PUS/ConvSsc/mininc. = STM35CubaDE File Edit Source Relator Navigate Search Project Run Window Help	- a ×
ne cas souce nextor hwydae secon priyect nun window nep ¹ / ₂ O Montanico (netr X	0 0 0 x x = e
and a strate of the strate of	
Stat a grout Constrained Constraine Constrained Constrained Cons	Support & Community Trailing Francisco ST Factor ST

or select Import item from File menu:



then choose Projects from Folder or Archive option from the Import dialog and press Next



elect an import wi	zard:		
ype filter text			
File Syst Import a Import a Import A Import S Preferen	Projects into Workspace em c6 System Workbench for STM32 Project n Existing STM32CubeMX Configuration File (.io tollic TrueSTUDIO Project TM32Cube Example ces from Folder or Archive	c)	~

In the following dialog select the folder from which import the existing project and press *Finish*:

Import Projects from	File System or Archive			×
	File System or Archive content of your folder or archive file to find projects and import them in the IDE.			7
Import source: C:\D/	TA\WorkSpaces\TestCM4 v	Directory	Archive	
type filter text		Select	All	
Folder	Import as	Deselec	t All	
TestCM4 TestCM4\CM0P TestCM4\CM0P TestCM4\CM4	US Eclipse project Eclipse project Eclipse project	3 of 3 selected	pen project	ts
Close newly importe	d projects upon completion			
Use installed project co				
Search for nested pr	-			
Working sets				
Add project to wo	rking sets		Vew	
Working sets:		~ 5	elect	
	<u>51</u>	how other specialized in	mport wizar	<u>ds</u>
?	< Back Next >	Finish	Cancel	



When importing a multiple project the user may choose to import the full project containing both the sub-projects for arm[®]Cortex[®]-M0+ and arm[®]Cortex[®]-M4 cores or simply the single project for the arm[®]Cortex[®]-M4 core.

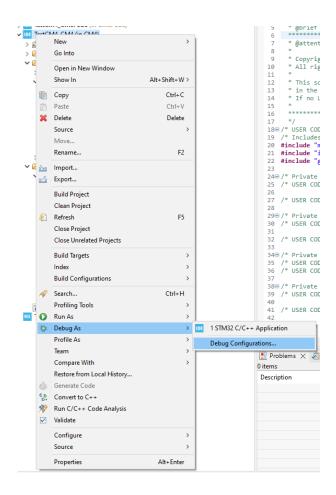
In the project import case there is no need of the arm[®]Cortex[®]-M0+ project because here the assumption is that the arm[®]Cortex[®]-M4 project was previously created considering the dual core operation scenario.

In the above example the previously created TestCM4 project has been imported but, of course, any existing project can be imported from its specific folder or archive in the same way.

6. Debugging STM32 arm®Cortex®-M4 code

The arm[®]Cortex[®]-M0+ code is protected and not accessible by debug tools. So the only debuggable part is the user code developed for the arm[®]Cortex[®]-M4 core.

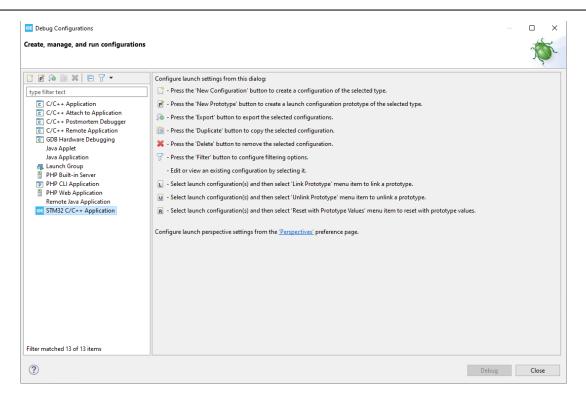
Before starting a debug session it is required to successfully build almost once the project and create a debug configuration. To do so right click on the project name in *Project Explorer* tab, select *Debug As* and then *Debug Configurations...*. Otherwise press the *Debug* icon and select *Debug Configurations...*.



蓉	• 🜔 • 🚱 • 隆 • 🌘	1
	(no launch history)	ā
	Debug As	>
	Debug Configurations	
	Organize Favorites	

In the Debug Configuration dialog double click on STM32 C/C++ Application





If the project had been successfully built almost once, a new debug configuration will be created with reference to the correct project:

Debug Configurations			-	
Create, manage, and run configurations				Ť.
Image: Second Secon	Name: TestCM4_CM4 Debug Main \$\$ Debugger Startup Source Project: TestCM4_CM4 C/C++ Application: Debug/TestCM4_CM4.elf Build (if required) before launching Build Configuration: Select Automatically O Enable auto build © Use workspace settings	Common	Search Project	Browse Browse V
Filter matched 14 of 15 items			THE TELE	
?			Debug	Close

If ST-LINK is used as debug device there is no need of further settings. Otherwise the debug device has to be set into *Debugger* tab:



Debug Configurations reate, manage, and run configuration	ns – □ >
Image: Construction C/C++ Application C/C++ Attach to Application C/C++ Application C/C++ Remote Application C/C++ Remote Application C/C++ Remote Application G/DB Hardware Debugging Java Applet Java Applet Java Applet PHP Buiktin Server STM32 C/C++ Application Image: TestCM4_CM4 Debug	Name TestCM4_CM4 Debug Image: TestCM4_CM4 Debugger Startup % Source Common GDB Connection Settings Image: TestCM4_CM4 Debugger Image: TestCM4_CM4 Debugger Startup % Source Common GDB Connection Settings Image: TestCM4_CM4 Debugger Image: TestCM4_CM4 Debugger Startup % Source Common GDB Server Host name or IP address Image: TestCM4_CM4 Debugger Startup % Source Common GDB Server Command Line Options Show Command Line Options Show Command Line Options Scan Frequency (kH2): Auto Access port: 0 - Cortex - M4 Image: Connect under reset Imable Debug in low power modes: Imable Supend watchdog counters while halted: No configuration Cross Trigger Interface (CTI) Imable
Filter matched 14 of 15 items	Never Apply

Pressing the *Debug* button the debug session will start and all classic debug tools will be available:

:t Ru	n Window Help					
3. 🖷		🎽 🔊 🖋 ㅋ 티네 - 씨티 및 ㅋ 🎘 ㅋ 박다 다 수 ㅋ 다 ㅋ 🛛 🚺			Q	😰 🖬 🔤 🎋
i⇒	Instruction Stepping Mode	TestCM4.ioc 🖸 main.c 🗙 🗖 🗖	(x)= Va × 💁 Br	. 🍕 Ex 🐮 In 🚟 F	le 65° Li 🖷	🖷 SF 😐 🗖
.3.	Move to Line (C/C++)	71				⊟ 📑 🛃 🕴
.76	Resume at Line (C/C++)	72 /* Reset of all peripherals, Initializes the Flash interface and the <u>Systick</u> . */ 73 HAL Init();	Name	Туре		lue
80	Resume F8	73 HAL_INIT(); 74		1964		
	Suspend	75 /* USER CODE BEGIN Init */				
	Terminate Ctrl+F2	76 77 /* USER CODE END Init */				
34	Disconnect	78				
÷.	Resume Without Signal	<pre>79 /* Configure the system clock */ 8 80 SystemClock Config();</pre>				
	Restart History	81				
	Restart	82 /* IPCC initialisation */ 83 MX IPCC Init();				
<u> </u>	Restart Configurations	84				
_	-	85 /* USER CODE BEGIN SysInit */ 86				
	Step Into F5 Step Over F6	87 /* USER CODE END SysInit */				
3		88 89 /* Initialize all configured peripherals */				
	Step Return F7	90 MX_GPIO_Init();				
	Run to Line Ctrl+R	91 /* USER CODE BEGIN 2 */				
	Use Step Filters	92 93 /* USER CODE END 2 */				
	Step Into Selection Ctrl+F5	94				
Q	Run	95 /* Boot CPU2 */ 96 HAL_PWREx_ReleaseCore(PWR_CORE_CPU2);				
枝	Debug F11	97				
	Run History	98 /* Infinite loop */ 99 /* USER CODE BEGIN WHILE */				
0	Run As	100 while (1)				
	Run Configurations	101 { 102 /* USER CODE END WHILE */	<			>
		103				^
	Debug History	104 /* USER CODE BEGIN 3 */ 105 }				
10	bebught	105 /* USER CODE END 3 */				
	Debug Configurations	107 }				
	Breakpoint Types	1090 /**				
0	Toggle Breakpoint Ctrl+Shift+B	110 * @brief System Clock Configuration 111 * Øretval None				
0	Toggle Line Breakpoint	112 */				
	Toggle Watchpoint	1130 void SystemClock_Config(void) 114 { v				
0	Toggle Method Breakpoint	114 {	<			>
6	Skip All Breakpoints Ctrl+Alt+B	👳 Console 🗙 👔 Problems 🕡 Executables 🗔 Debug Output 🗔 Browser Output 😨 Debugger Console 📋 Memory	-	XX 🕺 🖻 🖬 🕑 🚺		
8		TestCM4_CM4 Debug [STM32 C/C++ Application] [pid: 7]	-			· 🛃 • 🗳
20		researces and terms every white and the st				^
	Script Watch Expression Script Inspect					
		Verifying				
	Script Display					
	Inspect					
	Watch	Download verified successfully				
Q	External Tools	Domizous verzrzes successionaly				
_						



7. Flash memory and RAM considerations

Flash memory and RAM memory are shared resources but they are sharply splitted between cores to avoid situations where a core overflows to the memory domain assigned to the other one.

There is only one exception to this rule regarding the IPCC shared memory area. It is a 1 kB RAM area residing into arm[®]Cortex[®]-M0+ volatile memory zone but shared with arm[®]Cortex[®]-M4 to allow data exchange between cores.

The amount of flash memory available for the arm[®]Cortex[®]-M4 application code depends on the radio protocol stack chosen for the radio part. There are 5 available radio protocol stacks corresponding to as many software versions of the 32001506xxx module. They are listed in the following table:

Module type	Radio stack		Flash end address	Size
32001506Axx	WMBus		0x08031FFF	200 kB
32001506Bxx	LoRaWAN [®]		0x08029FFF	168 kB
32001506Cxx	LoRa [®] Mipot	0x08000000	0x0802C7FF	178 kB
32001506Dxx	LoRa [®] Modem	0x08000000 0x0802FFFF		192 kB
32001506Fxx	Multiprotocol LoRaWAN [®] + LoRa [®] Modem		0x08021FFF	136 kB

The amount of RAM memory available for the arm[®]Cortex[®]-M4 is the same for all module types as shown in the following table:

Module type	Radio stack	RAM start address	RAM end address	Size
ALL	ALL	0x20000000	0x20007FFF	32 kB

The IPCC shared RAM area is the same for all module types as shown in the following table:

Module type	Radio stack	IPCC start address	IPCC end address	Size
ALL	ALL	0x20008000	0x200083FF	1 kB

Flash and RAM memory areas are defined inside the linker file created for the specific build configuration used. As an example the linker file section defining the memory areas for the 32001506DEU is shown in the following image:



TestCM4.ioc 🙀 "STM32WL55J.CIX_FLASH.id ×	
	_
1/ ⁷ 2 ^{**} LinkerScript	^
2 * Linkerscript	
5 ° / 4	
4 + 5/* Entry Point */	
6 ENTRY(Reset_Handler) 7	
8/* Highest address of the user mode stack */ 9 estack = ORIGIN(RAM) + LENGTH(RAM); /* end of "SRAMI" Ram type memory */	
Descate - Uniting (whith + Length (whith,),) end of Shari Kain type memory /	
11 Min Heap Size = 0x200; /* required amount of heap */	
12 Min Stack Size = 0x400; / required amount of stack */	
13	
14/* Memories definition */	
15 MEMORY	
16 {	
17 ROM (rx) : ORIGIN = 0x080000000, LENGTH = 192K /* Flash memory dedicated to CM4 */	
18 RAM (xrw) : ORIGIN = 0x20000000, LENGTH = 32K /* Non-backup SRAM1 dedicated to CM4 */	
19 }	
20	
21/* Sections */	
22 SECTIONS	
23 {	
24 /* The startup code into "ROM" Rom type memory */	
25 .isr_vector :	
26 {	
27 . = ALIGN(8);	
<pre>28 KEEP(*(.isr_vector)) /* Startup code */</pre>	
29 . = ALIGN(8);	
30 } >ROM	
31	
32 /* The program code and other data into "ROM" Rom type memory */	
33 .text :	
34 {	
35 . = ALIGN(8);	
36 *(.text) /* .text sections (code) */	
37 *(.text*) /* .text* sections (code) */	
38 *(.glue_7) /* glue arm to thumb code */	
39 *(.glue_7t) /* glue thumb to arm code */	
40 *(.eh_frame)	
41	
42 KEEP (*.init))	
43 KEEP (*(.fini))	
44	
45 .= ALIGN(8);	
46 /* define a global symbols at end of code */	
47 } >ROM	
49 /* Constant data into "ROM" Rom type memory */	
50 .rodata :	
51 {	~
52 . = ALIGN(8);	1
	_
STM32WL55JCIX_FLASH.Id	



8. STM32 arm[®]Cortex[®]-M0+ boot

The arm[®]Cortex[®]-M0+ can boot only if the arm[®]Cortex[®]-M4 enables it by setting the bit 15 *C2BOOT* (*CPU2 boot after reset or wakeup from Stop or Standby mode*) of the PWR control register 4 (*PWR_CR4*).

This can be done calling the function HAL_PWREx_ReleaseCore(PWR_CORE_CPU2) if using HAL drivers, calling the function LL_PWR_EnableBootC2() if using LL drivers or directly writing to the PWR_CR4 register (SET_BIT(PWR->CR4, PWR_CR4_C2BOOT)).

In the following image the Cpu2 boot is performed using the HAL function call.

⊖int main(void)
{ /* USER CODE BEGIN 1 */
/* USER CODE END 1 */
/* MCU Configuration*/
/* Reset of all peripherals, Initializes the Flash interface and the $\underline{Systick}.$ */ HAL_Init();
/* USER CODE BEGIN Init */
/* USER CODE END Init */
<pre>/* Configure the system clock */ SystemClock_Config();</pre>
<pre>/* IPCC initialisation */ MX_IPCC_Init();</pre>
/* USER CODE BEGIN SysInit */
/* USER CODE END SysInit */
/* Initialize all configured peripherals */ MX_GPIO_Init(); /* USER CODE BEGIN 2 */
/* USER CODE END 2 */
<pre>/* Boot CPU2 */ HAL_PWREx_ReleaseCore(PWR_CORE_CPU2);</pre>
/* Infinite loop */ /* USER CODE BEGIN WHILE */ while (1) { /* USER CODE END WHILE */
/* USER CODE BEGIN 3 */ } /* USER CODE END 3 */ }



9. Inter-core communication

Inter-core communication is performed through IPCC (Inter-Processor Communication Controller) which is based on the 1 kB shared volatile memory area physically located at the beginning of the arm[®]Cortex[®]-M0+ RAM domain.

Communication is logically separated into two kinds of messages: main communication protocol messages used for radio module control (e.g. working parameters configuration, transmit and receive commands) and internal service messages used for inter-processor coordination (e.g. low power management).

Messages are internally routed through different physical channels; so a communication channel and a service channel are implemented, each of them based on a specific communication buffer residing at different locations into shared memory.

There is also a further shared memory location used for extra-IPCC information exchange; it is used e.g. to share the arm[®]Cortex[®]-M0+ status (started or not) or to transfer to the arm[®]Cortex[®]-M0+ data or commands asynchronous with respect to the normal IPCC flow (e.g. a core reset request).

RAM RAM Size **Buffer type** end Usage start address address Communication protocol Communication 0x200082FC 0x200083FF 260 Bytes messages Service Internal service messages 0x200082EC 0x200082FB 16 Bytes Extra-data Asynchronous messages 0x200082E8 0x200082EB 4 Bytes

The above mentioned buffers are listed in the following table:

9.1. Communication messages

The communication over the communication channel between the arm[®]Cortex[®]-M4 and arm[®]Cortex[®]-M0+ cores is performed with the same communication protocol defined for the physical SPI/I²C/UART channels used into 32001505xxx. For details please refer to *"32001505xxx_Command_Reference_revx.x.pdf"* document.

The core that initiates a data exchange over the communication channel must:

- verify that the channel is free;
- write the protocol message into the communication buffer;
- notify to the other core that the channel is busy and a message is waiting to be read;
- wait for the receiving core to notify that the message has been read and the channel is free again.

The receiving core can react in two ways:

- if no reply is needed for the received message the core clears the communication buffer and simply notifies that the channel is free;
- if a reply is required the core must clear the communication buffer, write its answer into the communication buffer and then notify that the channel is free.



The sending core, on its side, when receives a channel free notify must always check the communication buffer for a valid message present. No further notification has to be sent and the IPCC communication cycle is anyway considered to be completed.

It is mandatory to perform the complete IPCC communication cycle for every sent message otherwise the inter-core communication stops and waits for the missing step.

9.2. Service messages

The communication over the service channel between the arm[®]Cortex[®]-M4 and the arm[®]Cortex[®]-M0+ cores is based on a very small set of commands using the following minimal protocol syntax:

```
<cmd>[<arg>]
```

where

- <*cmd*> is the command code;
- *<arg>* is the optional argument associated to the command.

Both <*cmd*> and <*arg*> are 1 byte fields.

Service commands are listed in the following table:

Command code	Argument	Description	Direction
0x01	none	Request low power mode	СМ4 ←→ СМ0+
0x02	none	Request wake-up from low power	СМ4 ←→ СМ0+
0x05	baud rate	Request UART baud rate setting	СМ4 ← СМ0+
0xA5	none	End of message processing	CM4 → CM0+

Commands 0x01 'Request low power mode' and 0x02 'Request wake-up from low power' can be sent from both cores depending on the module's working mode. Please refer to paragraph "11. Module's pins usage" to gather more information about how module pins are managed.

If the module uses pins to change and signal its status then commands 0x01 and 0x02 are sent from arm[®]Cortex[®]-M0+ to arm[®]Cortex[®]-M4 because in this case low power is managed by arm[®]Cortex[®]-M0+ and propagated to arm[®]Cortex[®]-M4.

Otherwise low power is managed by arm[®]Cortex[®]-M4 and propagated to arm[®]Cortex[®]-M0+ through commands 0x01 and 0x02, so they follow the opposite direction.

Command *0x05 'Request UART baud rate setting'* is used only in the case of stand-alone module emulation mode with UART communication. In this working mode the module behaves as a stand-alone module with the arm®Cortex®-M4 routing messages from the physical UART to the arm®Cortex®-M0+ core and vice versa. As the baud rate value is a parameter contained into the arm®Cortex®-M0+ configuration memory and the UART is managed by the arm®Cortex®-M4, the arm®Cortex®-M0+ uses this command to inform the arm®Cortex®-M4 about the baud rate to set. Admitted values for baud rate are:



Baud rate value	Baud	
0x00	9600	
0x01	19200	
0x02	38400	
0x03	57600	
0x04	115200	

Command *0xA5 'End of message processing'* is a sort of acknowledge that the arm[®]Cortex[®]-M4 sends to arm[®]Cortex[®]-M0+ after receiving a communication message from it. Its aim is to prevent the arm[®]Cortex[®]-M0+ sending further communication messages before the arm[®]Cortex[®]-M4 completes processing the previous one. arm[®]Cortex[®]-M4 must send it otherwise arm[®]Cortex[®]-M0+ will not send any new messages.

Service messages comply to the same IPCC rules already mentioned for communication messages, except for the fact that they use the service channel and the related buffer.

The core that initiates a data exchange over the service channel must:

- verify that the channel is free;
- write the message into the service buffer;
- notify to the other core that the channel is busy and a message is waiting to be read;
- wait for the receiving core to notify that the message has been read and the channel is free again.

On the service channel commands do not require a reply. So the receiving core reacts in the following way:

• the core clears the communication buffer and simply notifies that the channel is free; No further notification has to be sent and the IPCC communication cycle is anyway considered to be completed.

It is mandatory to perform the complete IPCC communication cycle for every sent message otherwise the inter-core communication stops and waits for the missing step.



9.3. Asynchronous messages

Asynchronous messages are commands or signals without arguments that are exchanged out of the regular IPCC communication flow. They are listed in the following table:

Command code	Description	Direction
0xAA	arm [®] Cortex [®] -M0+ not started	СМ4 → СМ0+
0xBB	arm [®] Cortex [®] -M0+ started	СМ4 ← СМ0+
0xCC	Request IPCC interface re-initialization	СМ4 → СМ0+
0xDD	Request sub-GHz radio re-initialization	СМ4 → СМ0+
OxEE	Request arm [®] Cortex [®] -M0+ restart	СМ4 → СМ0+



10. Low power management

Details about power management can be found into chapter 6 *Power control (PWR)* of the STM32WL55JC reference manual.

Here we just recap the concept that three power domains have to be considered: arm[®]Cortex[®]-M4 power domain, arm[®]Cortex[®]-M0+ power domain and System power domain. When talking about low power management it is necessary to clarify at which power domain it is referred to. It may happen that at a specific time arm[®]Cortex[®]-M4 core is in a particular low power state while arm[®]Cortex[®]-M0+ is in a different one. The System low power condition corresponds to the lowest (in terms of deepness in sleep condition) low power state between the ones of each core's power domain. If arm[®]Cortex[®]-M4 is in Sleep mode while arm[®]Cortex[®]-M0+ is in Stop 2 mode the System is considered to be in Sleep mode.

When arm®Cortex®-M0+ is set to low power it reaches Stop 2 mode.

Module 32001506xxx provides two ways to manage low power depending on the pin usage set during configuration.

If pin usage is enabled then the module activates signalling through pins and low power management is performed through NWAKE pin level checking:

- NWAKE pin high \rightarrow module to low power;
- NWAKE pin low \rightarrow wake up module.

NWAKE pin level checking is performed by arm[®]Cortex[®]-M0+ core. So when a change is detected the arm[®]Cortex[®]-M0+ behaves as follows:

- if NWAKE becomes high, it sends the service command 0x01 to arm[®]Cortex[®]-M4 to signal it that a low power request has arised and then goes to Stop 2 low power mode;
- if NWAKE becomes low, it wakes up from Stop 2 mode and then sends service command 0x02 to arm[®]Cortex[®]-M4 to signal it that a wake up request has arised.

If pin usage is disabled then low power management has to be performed by the user application controlling the arm[®]Cortex[®]-M4 core.

When the user application requests a low power condition it sends the service command 0x01 to arm[®]Cortex[®]-M0+ core which goes to Stop 2 low power mode.

When the user application requests a wake up from low power condition it sends the service command 0x02 to arm[®]Cortex[®]-M0+ core which wakes up from Stop 2 low power mode.



11. Module's pins usage

Module's pins usage is the feature that allows the arm[®]Cortex[®]-M0+ core to manage low power condition based on the level of NWAKE pin and to signal radio data availability through NDATA_INDICATE pin.

Pins usage is kept for compatibility with the stand-alone version of the module (32001505xxx product) but is normally disabled by default.

To enable pins usage the following configuration command has to be sent to the arm[®]Cortex[®]- M0+ core through the IPCC interface mechanism:

0xAA 0x32 0x02 0x91 0x01 0x90

To disable pins usage the command is:

0xAA 0x32 0x02 0x91 0x00 0x91

To read the pins usage status issue the command:

0xAA 0x33 0x01 0x91 0x91

Above commands correspond to configuration memory write/read commands at address 0x91 which is the location of the parameter that defines the pins usage. Admitted values are $0x00 \rightarrow pins$ disabled and $0x01 \rightarrow pins$ enabled.



12. Revision History

Revision	Date	Description
0.1	08.11.2022	Preliminary