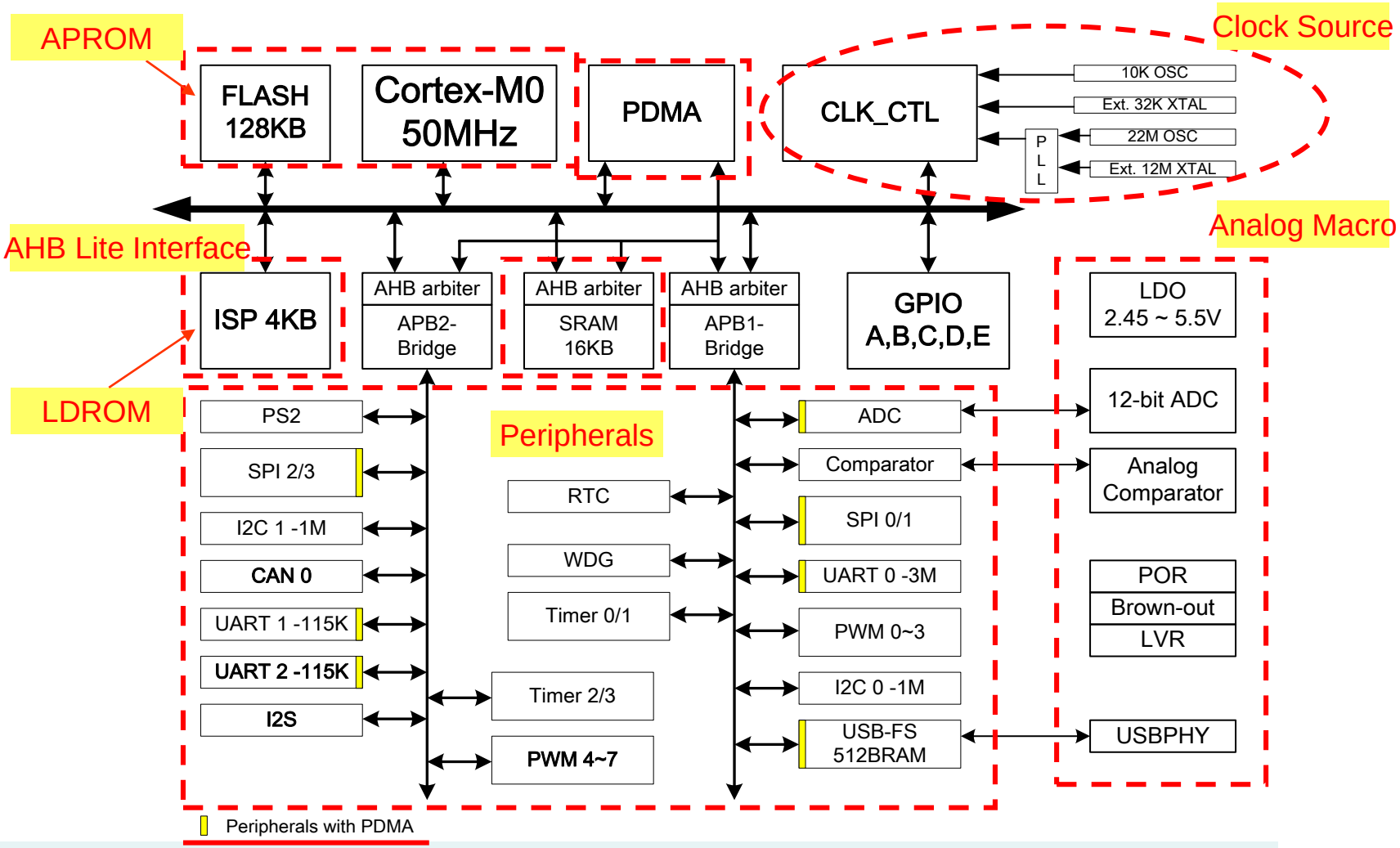
NuMicro CMSIS

- 3/8/2012

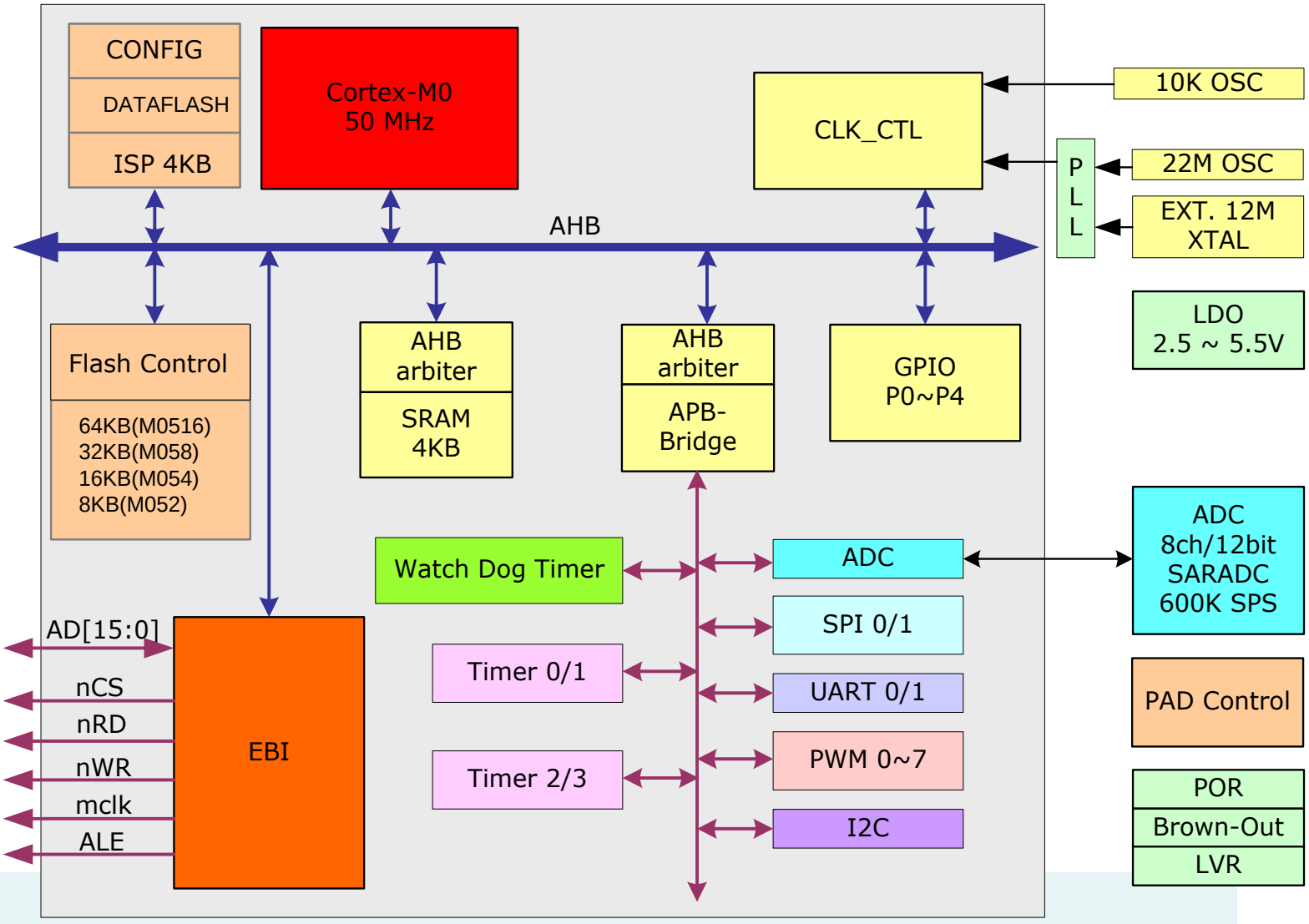
Agenda

- Block Diagram
- ISP/ICP
- System Memory Map
- Power Management
- Reset Source
- Clock Control
- Interrupt
- CMSIS

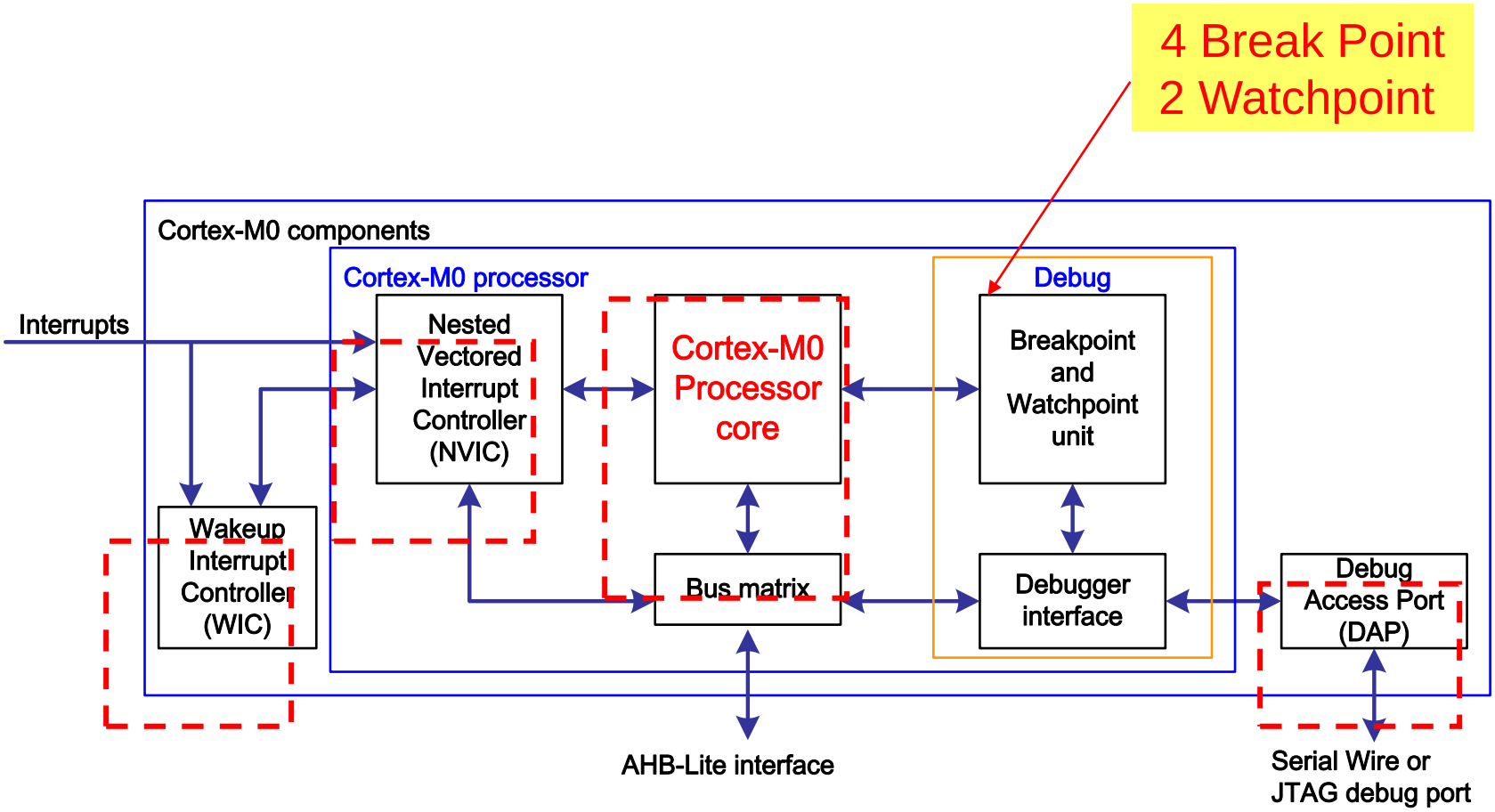
NUC100 Series Functional Block Diagram



M051 Series Functional Block Diagram



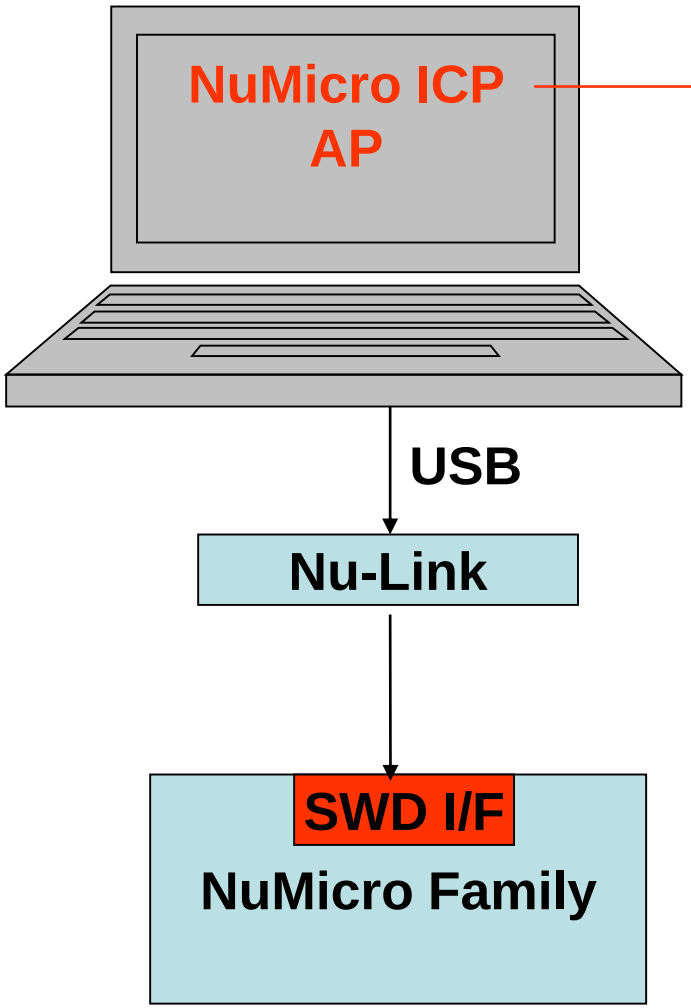
ARM Cortex-M0 Functional Block Diagram

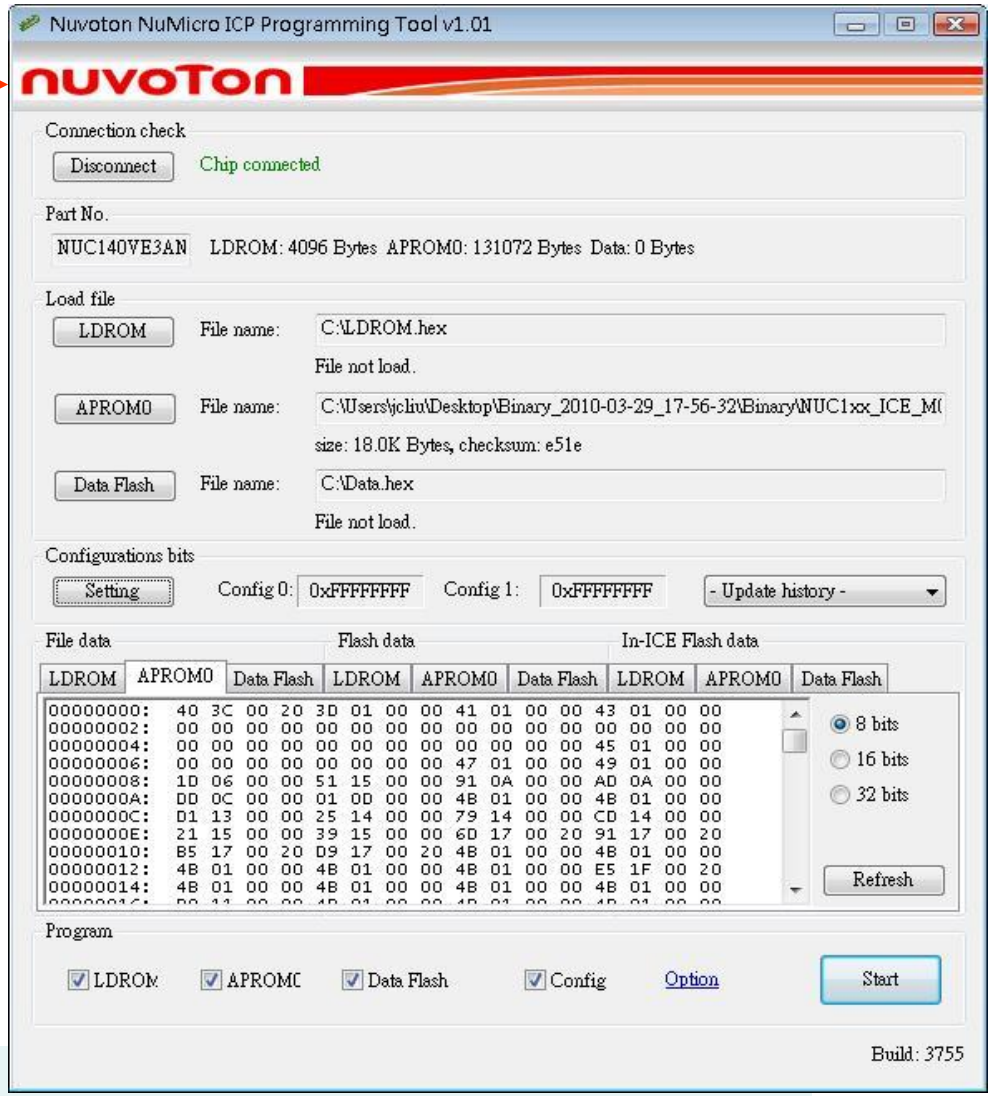


NuMicro Family System Introduction

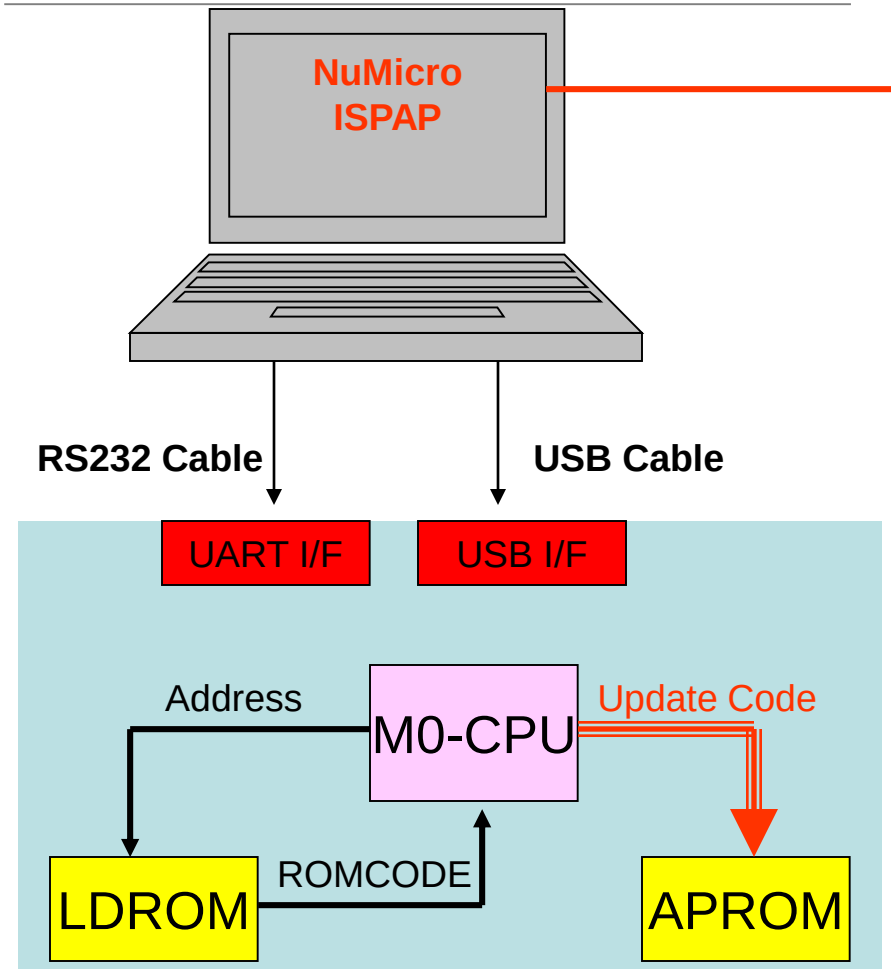
- The NUC 100 series IC embed Cortex –M0 core
  - running up to 50 MHz
  - 32K/64K/128K bytes embedded flash by part No
    - Configurable data flash address and size for 128kB system, fixed 4kB data flash for the 32kB and 64kB system
  - 4kB flash for ISP loader
  - 4K/8K/16K embedded SRAM
- The M051 series IC embed Cortex-M0 core
  - 8KB/16KB/32KB/64KB Flash memory for program memory (APROM)
  - 4KB Flash memory for data memory (DataFlash)
  - 4KB Flash memory for loader (LDROM)
  - 4KB SRAM for internal scratch-pad RAM (SRAM)
- Support ISP (In System Programming) by USB or UART
- Support 2 wire ICP (In Circuit Programming) update from ICE interface
- Support fast parallel programming mode by external writer
- 2 wire SWD ICE interface
- Wide operating voltage ranges from 2.5V to 5.5V

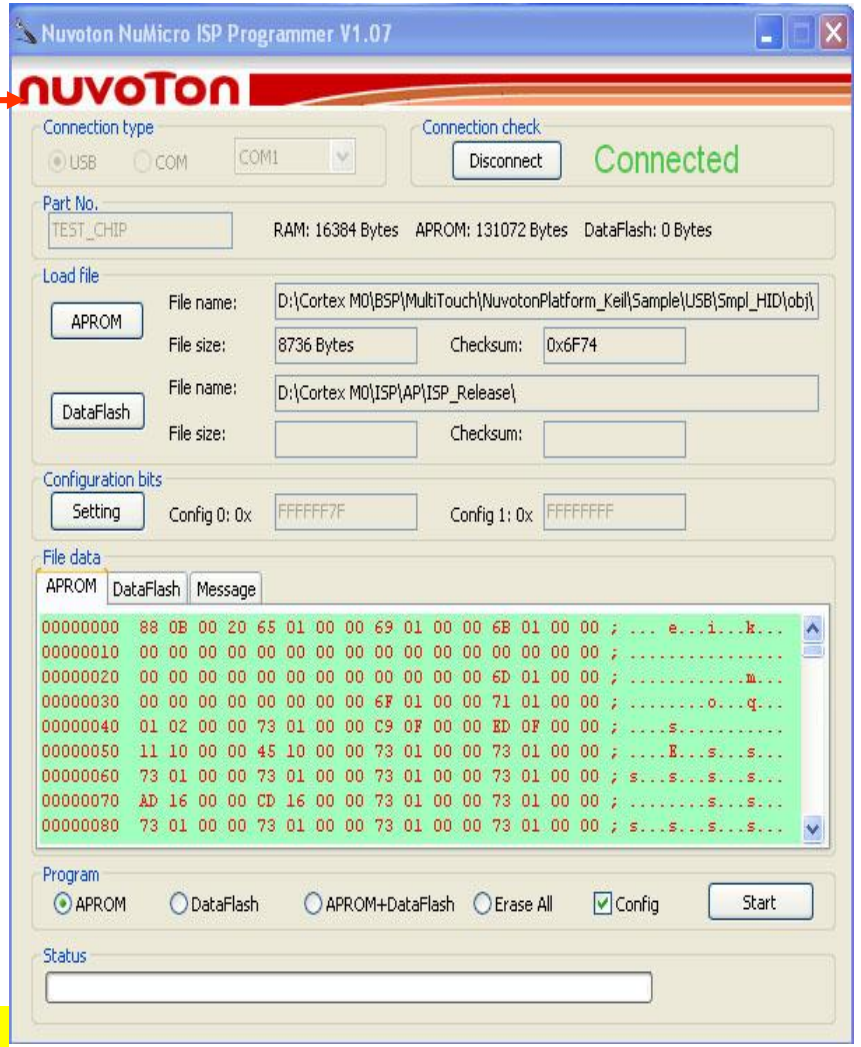
ICP (In Circuit Programming)





ISP (In System Programming)



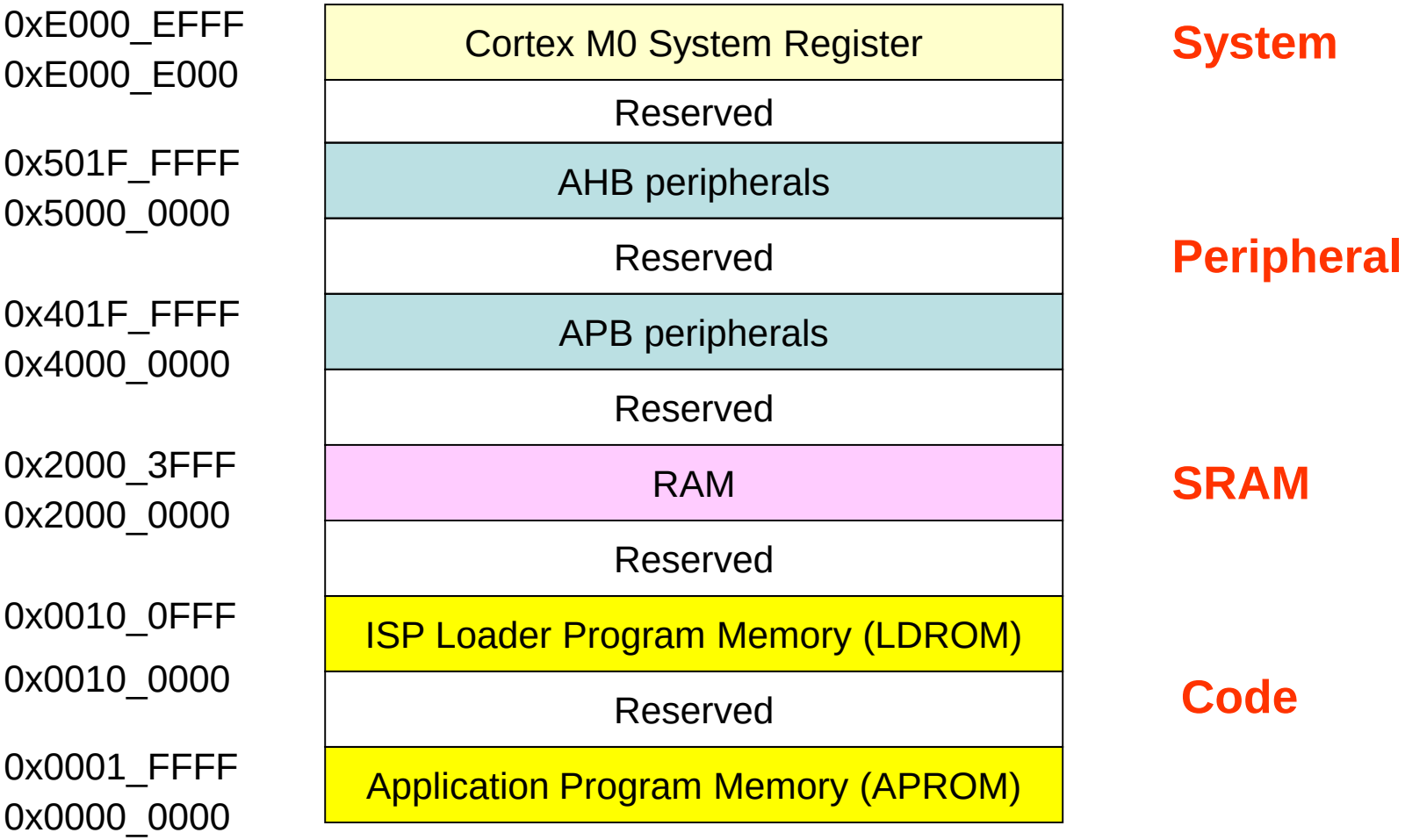


- Nuvoton ISP LDROM firmware code
- Boot From LDROM
- USB interface only supports NUC120/NUC140 series

NuMicro ISP vs ICP Difference

Item	ISP	ICP
PC AP Software	NuMicro ISP Programmer	NuMicro ICP Programming Tools.exe
Extra Hardware	No	NuLink
CPU Run Code	CPU runs on the LDROM	No
What Region Can Be Updated	•APROM •DataFlash •Config	•APROM •LDROM •DataFlash •Config
Interface	USB or UART	SWD
Support Off-Line Programming	No	Yes

System Memory Map



Power Management

- **Normal Run Mode**
  - Flexible system clock source selection
  - All peripherals clock can be turned off individually.
- **Sleep Mode (IDLE Mode)**
  - CPU halt, peripheral is probably under running which depends on your application
- **Deep Sleep Mode (Power Down Mode)**
  - CPU & peripheral are all halt

Power Management – Cont.

- Sleep mode

```
UNLOCKREG();
SCB->SCR = 4;
SYSCLK->PWRCON.WINT_EN     = 0;
SYSCLK->PWRCON.PD_WAIT_CPU = 1;
SYSCLK->PWRCON.PWR_DOWN    = 0;
LOCKREG();
__WFI();
```

- Deep Sleep mode

```
UNLOCKREG();
SCB->SCR = 4;
SYSCLK->PWRCON.WINT_EN     = 0;
SYSCLK->PWRCON.PD_WAIT_CPU = 1;
SYSCLK->PWRCON.PWR_DOWN    = 1;
LOCKREG();
__WFI();
```

System Reset

- **Hardware Reset**
  - The Power-On Reset
  - /RESET pin Reset
  - Watchdog Time Out Reset
  - Low Voltage Detected Reset (generates a reset when Vcc < Vref.)
  - Brown-Out-Detected Reset (an intentional or unintentional drop in voltage)
- **Software Reset**
  - CPU_RST
    - To write 1 to the CPU_RST(IPRSTC1[1],"IP Reset Source Register") register
    - Just only reset CPU & flash controller
  - CHIP_RST
    - To write 1 to CHIP_RST(IPRSTC1[0], IP Reset Source Register) register
    - To reset the whole chip like "Power-on reset"
  - MCU_RST
    - Write 1 to the SYSRESETREQ (AIRCR[2], Application Interrupt & Reset Control Register")
    - To reset the whole chip

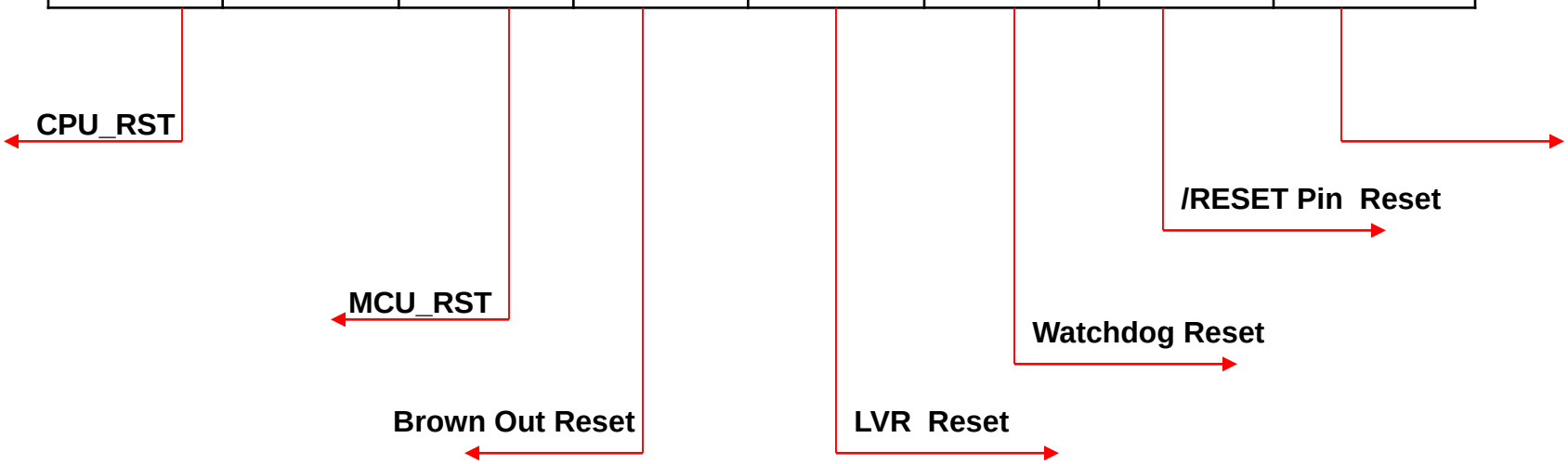
"RSTSRC" register identify chip's reset source from last operation

RSTRC(System Reset Source Register)

Register	Address	R/W	Description	Reset Value
RSTSRC	0x5000_0004	R/W	System Reset Source Register	0x0000_00xx

These bits are cleared by writing 1 to itself

7	6	5	4	3	2	1	0
RSTS_CPU	Reserved	RSTS_MCU	RSTS_BOD	RSTS_LVR	RSTS_WDG	RSTS_PAD	RSTS_POR



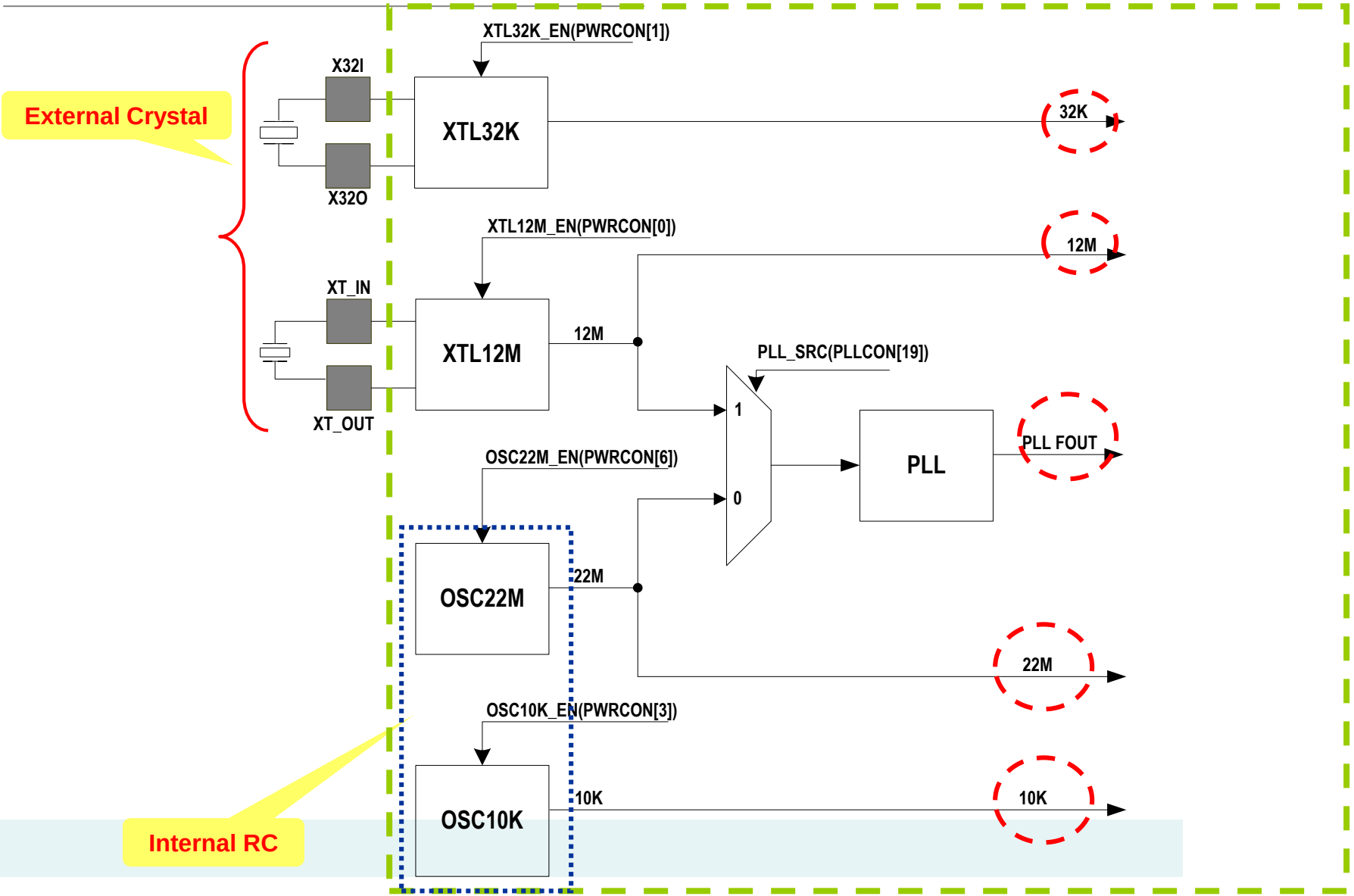
System Reset Resource Table

Reset Status Reset Source	7	6	5	4	3	2	1	0
	RSTS_CPU	Reserved	RSTS_MCU	RSTS_BOD	RSTS_LVR	RSTS_WDG	RSTS_PAD	RSTS_POR
Power-On Reset	0	x	0	0	0	0	1	1
Reset Pin Reset	0	x	0	0	0	0	1	0
Watchdog Time Out Reset	0	X	0	0	0	1	0	0
LVR Reset	0	X	0	0	1	0	0	0
Brown-Out Reset	0	X	0	1	0	0	0	0
Cortex-M0 MCU Reset	0	X	1	0	0	0	0	0
CHIP_RST	0	X	0	0	0	0	0	1
CPU_RST	1	X	0	0	0	0	0	0

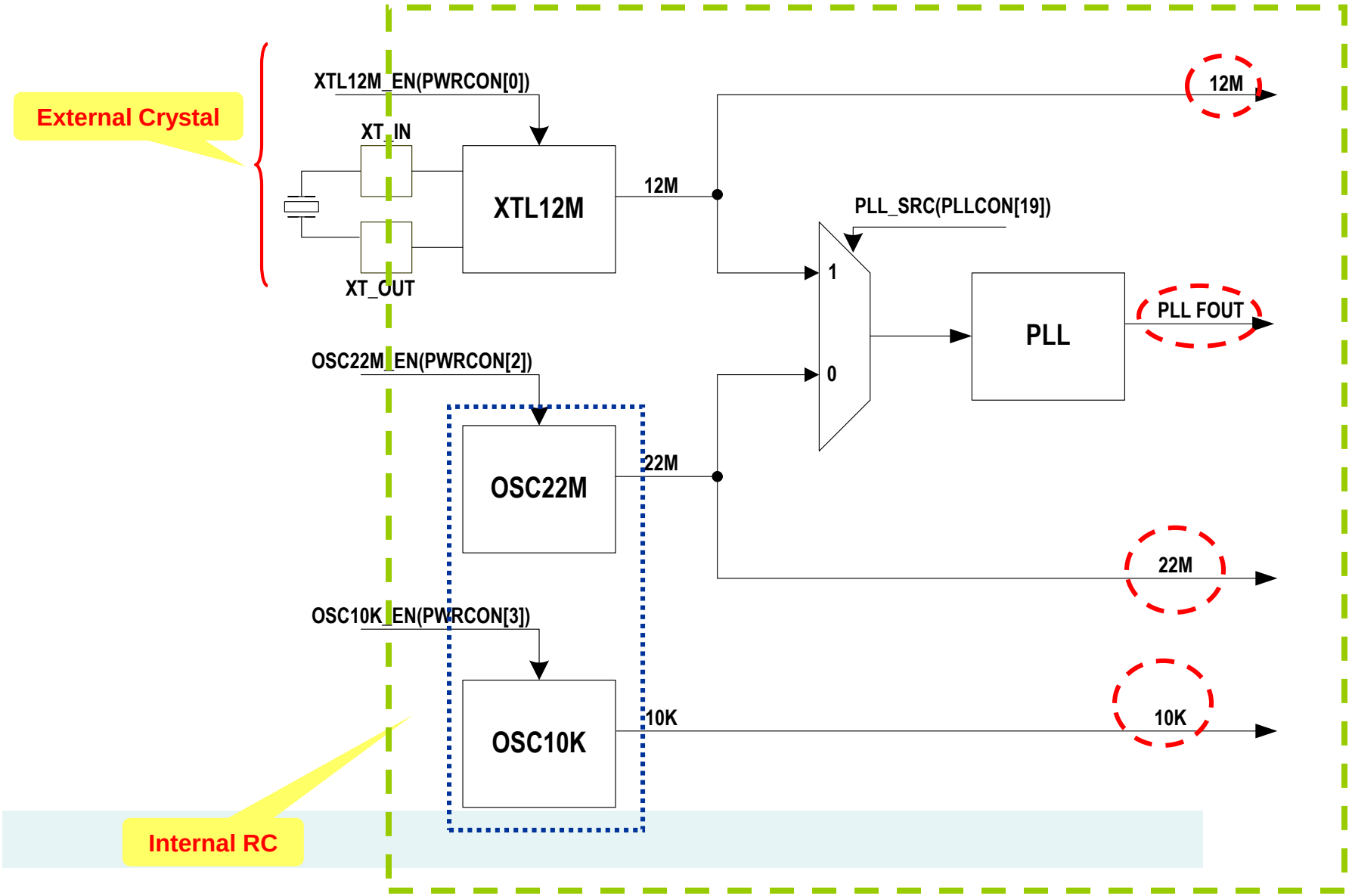
Peripheral IP Reset

- Every peripheral has corresponded reset register
- “IPRSTC1” & “IPRSTC2” register had defined the corresponded peripheral asynchronous reset signal

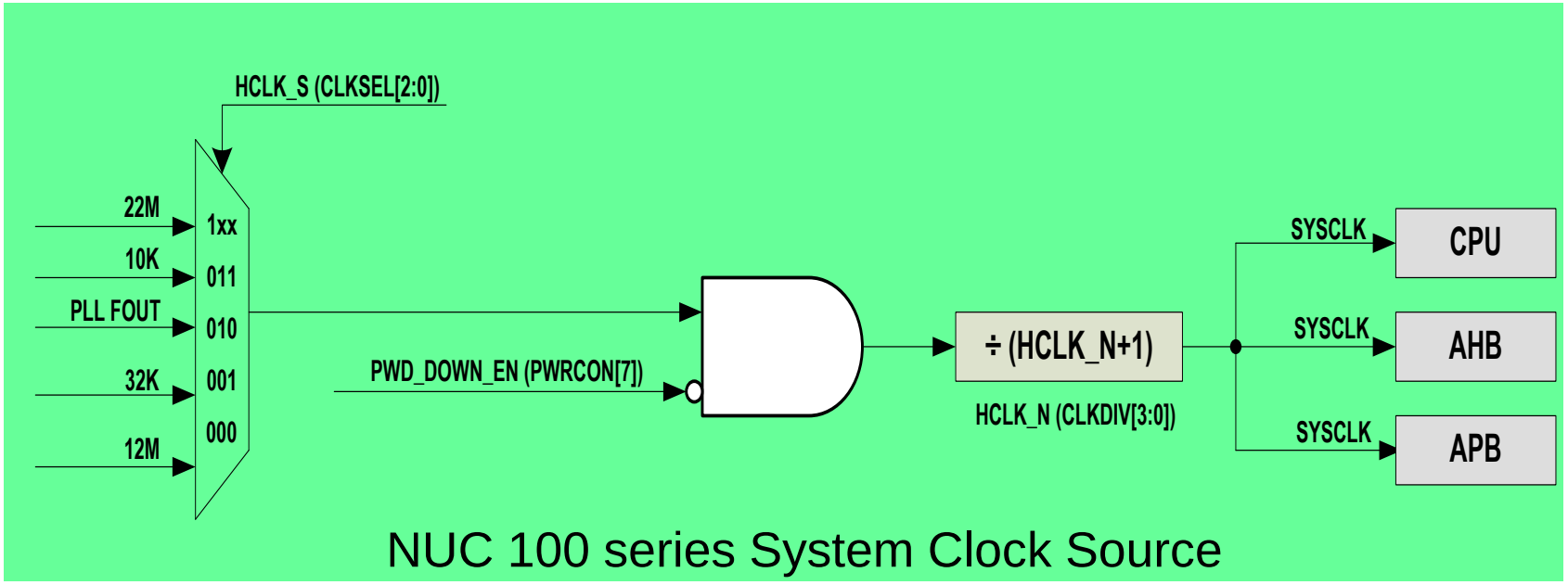
NUC100 Clock Generator



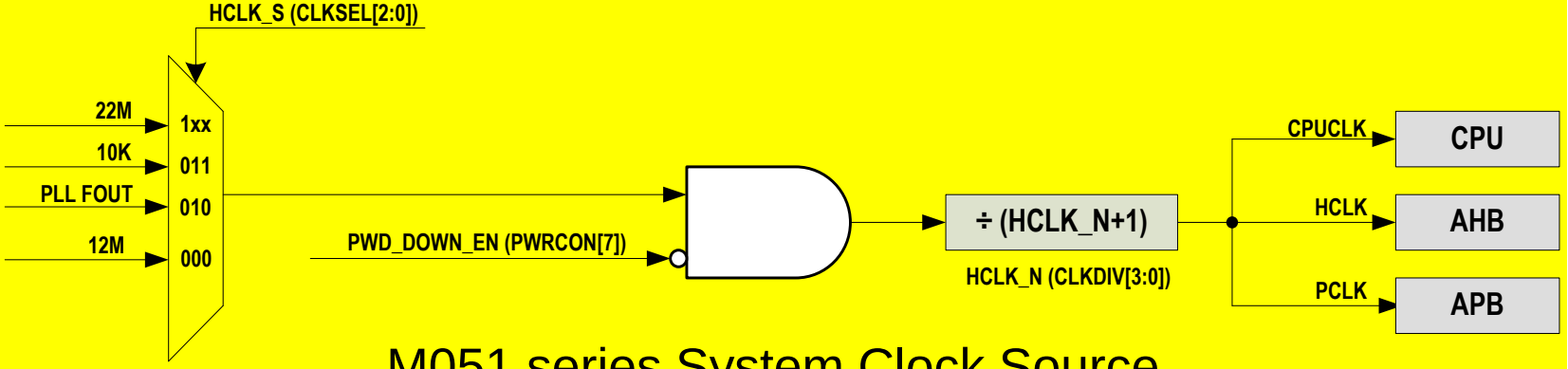
M051 Clock Generator



NuMicro System Clock Source

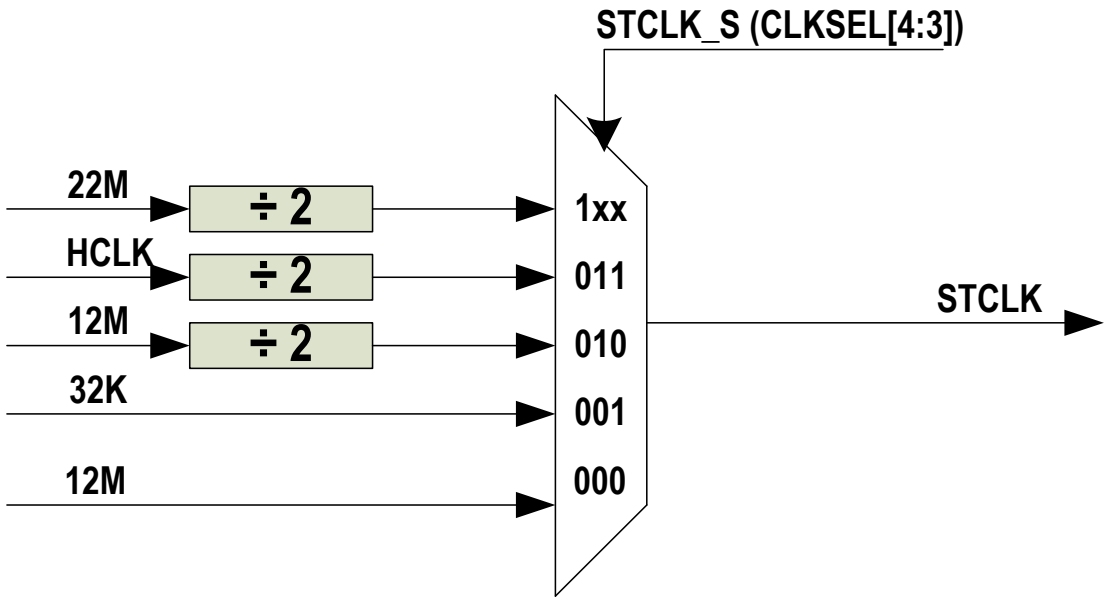


NUC 100 series System Clock Source



M051 series System Clock Source

SysTick Clock Source



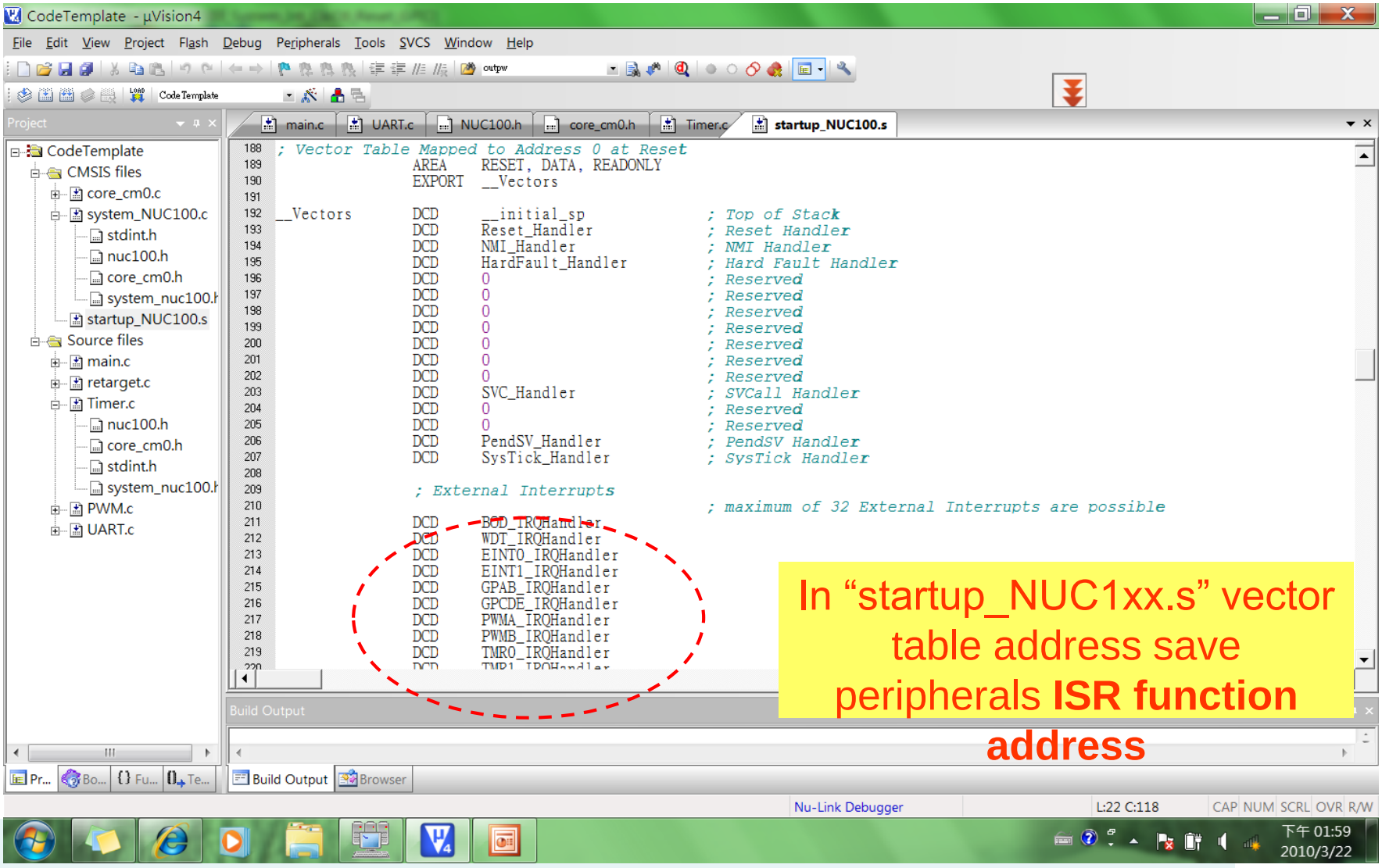
NUC100 Series System Clock Source

For the other peripherals IP, they also have multi-clock source to set.

NuMicro NVIC

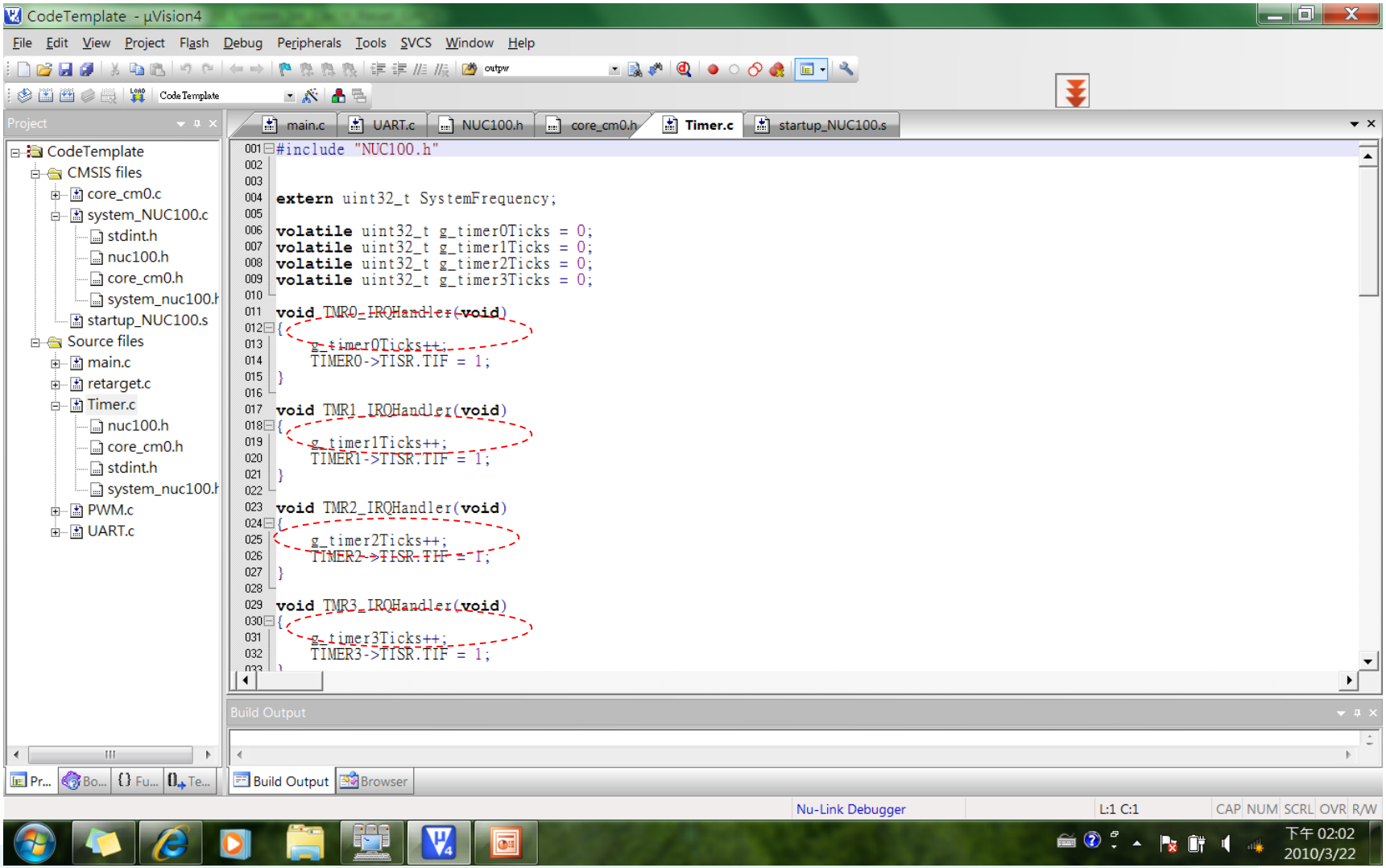
- NVIC (Nested Vectored Interrupt Controller)
  - An integrated part of the Cortex-M0 processor
  - It supports 32 peripheral interrupts input
  - it supports NMI(Nonmaskable Interrupt) input
  - It supports “Tail Chaining” & “Late Arrival”
- Interrupt handler follows the CMSIS coding rule

C ISR Function Call Name



In “startup_NUC1xx.s” vector table address save peripherals **ISR function address**

ISR Handler


CodeTemplate - µVision4
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Project
CodeTemplate
CMSIS files
core_cm0.c
system_NUC100.c
stdint.h
nuc100.h
core_cm0.h
system_nuc100.h
startup_NUC100.s
Source files
main.c
retarget.c
Timer.c
nuc100.h
core_cm0.h
stdint.h
system_nuc100.h
PWM.c
UART.c
main.c
UART.c
NUC100.h
core_cm0.h
Timer.c
startup_NUC100.s
#include "NUC100.h"
extern uint32_t SystemFrequency;
volatile uint32_t g_timer0Ticks = 0;
volatile uint32_t g_timer1Ticks = 0;
volatile uint32_t g_timer2Ticks = 0;
volatile uint32_t g_timer3Ticks = 0;
void TMR0_IRQHandler(void)
{
g_timer0Ticks++;
TIMER0->TISR.TIF = 1;
}
void TMR1_IRQHandler(void)
{
g_timer1Ticks++;
TIMER1->TISR.TIF = 1;
}
void TMR2_IRQHandler(void)
{
g_timer2Ticks++;
TIMER2->TISR.TIF = 1;
}
void TMR3_IRQHandler(void)
{
g_timer3Ticks++;
TIMER3->TISR.TIF = 1;
Build Output
Build Output
Browser
Nu-Link Debugger
L:1 C:1
CAP NUM SCRL OVR R/W
下午 02:02
2010/3/22

System Peripheral Interrupt Map(16~32)

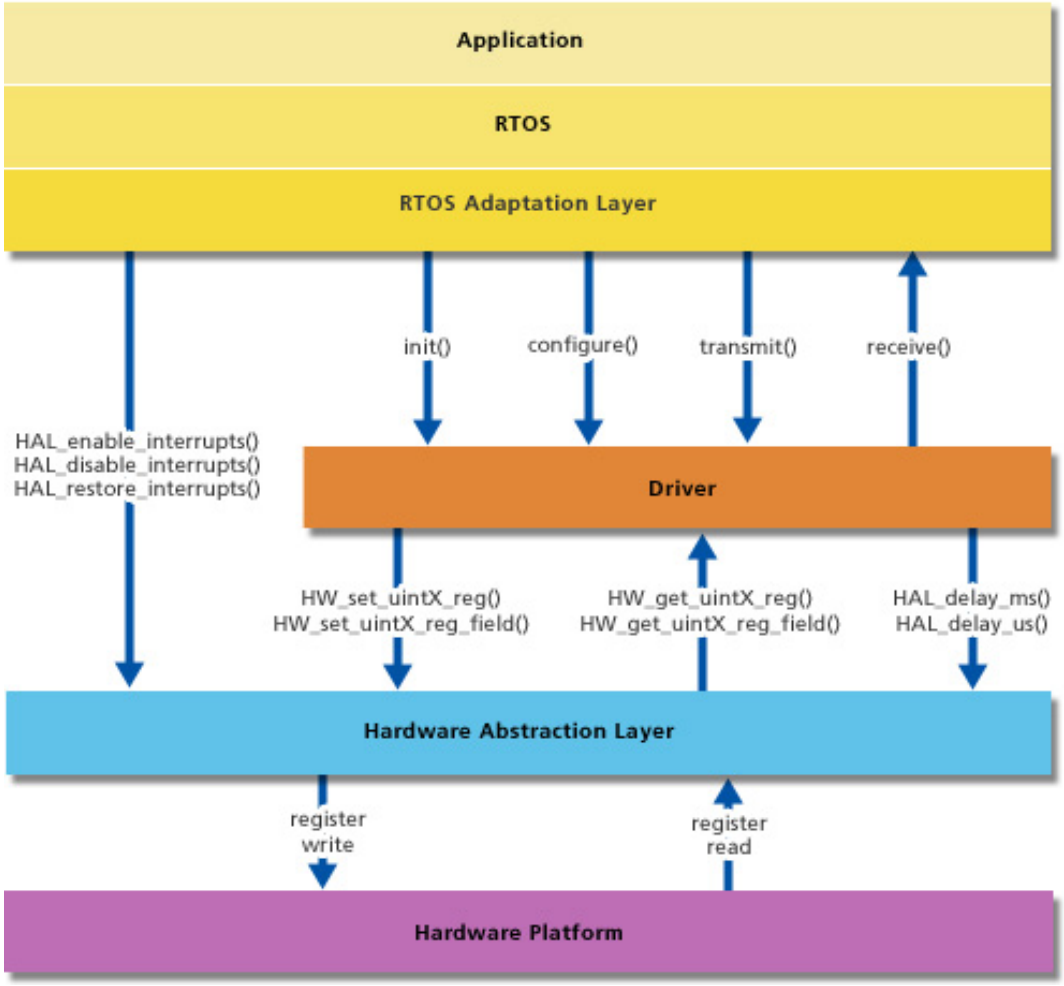
Vector Number	Interrupt Number (Bit in Interrupt Registers)	Interrupt Name	Source IP	Interrupt description
0 ~ 15	**-**	**-**	**-**	**System exceptions**
16	0	BOD_OUT	Brown-Out	Brownout low voltage detected interrupt
17	1	WDT_INT	WDT	Watch Dog Timer interrupt
18	2	EINT0	GPIO	External signal interrupt from PB.14 pin
19	3	EINT1	GPIO	External signal interrupt from PB.15 pin
20	4	GPAB_INT	GPIO	External signal interrupt from PA[15:0] / PB[13:0]
21	5	GPCDE_INT	GPIO	External interrupt from PC[15:0]/PD[15:0]/PE[15:0]
22	6	PWMA_INT	PWM0~3	PWM0, PWM1, PWM2 and PWM3 interrupt
23	7	PWMB_INT	PWM4~7	PWM4, PWM5, PWM6 and PWM7 interrupt
24	8	TMR0_INT	TMR0	Timer 0 interrupt
25	9	TMR1_INT	TMR1	Timer 1 interrupt
26	10	TMR2_INT	TMR2	Timer 2 interrupt
27	11	TMR3_INT	TMR3	Timer 3 interrupt
28	12	UART0_INT	UART0	UART0 interrupt
29	13	UART1_INT	UART1	UART1 interrupt
30	14	SPI0_INT	SPI0	SPI0 interrupt
31	15	SPI1_INT	SPI1	SPI1 interrupt
32	16	SPI2_INT	SPI2	SPI2 interrupt

System Peripheral Interrupt Map(33~47)

Vector Number	Interrupt Number (Bit in Interrupt Registers)	Interrupt Name	Source IP	Interrupt description
33	17	SPI3_INT	SPI3	SPI3 interrupt
34	18	I2C0_INT	I2C0	I2C0 interrupt
35	19	I2C1_INT	I2C1	I2C1 interrupt
36	20	CAN0_INT	CAN0	CAN0 interrupt
37	21	Reserved	Reserved	Reserved
38	22	Reserved	Reserved	Reserved
39	23	USB_INT	USBD	USB FS Device interrupt
40	24	PS2_INT	PS2	PS2 interrupt
41	25	ACMP_INT	ACMP	Analog Comparator-0 or Comaprator-1 interrupt
42	26	PDMA_INT	PDMA	PDMA interrupt
43	27	I2S_INT	I2S	I2S interrupt
44	28	PWRWU_INT	CLKC	Clock controller interrupt for chip wake up from power-down state
45	29	ADC_INT	ADC0/1	ADC interrupt
46	30	Reserved	Reserved	Reserved
47	31	RTC_INT	RTC	Real time clock interrupt

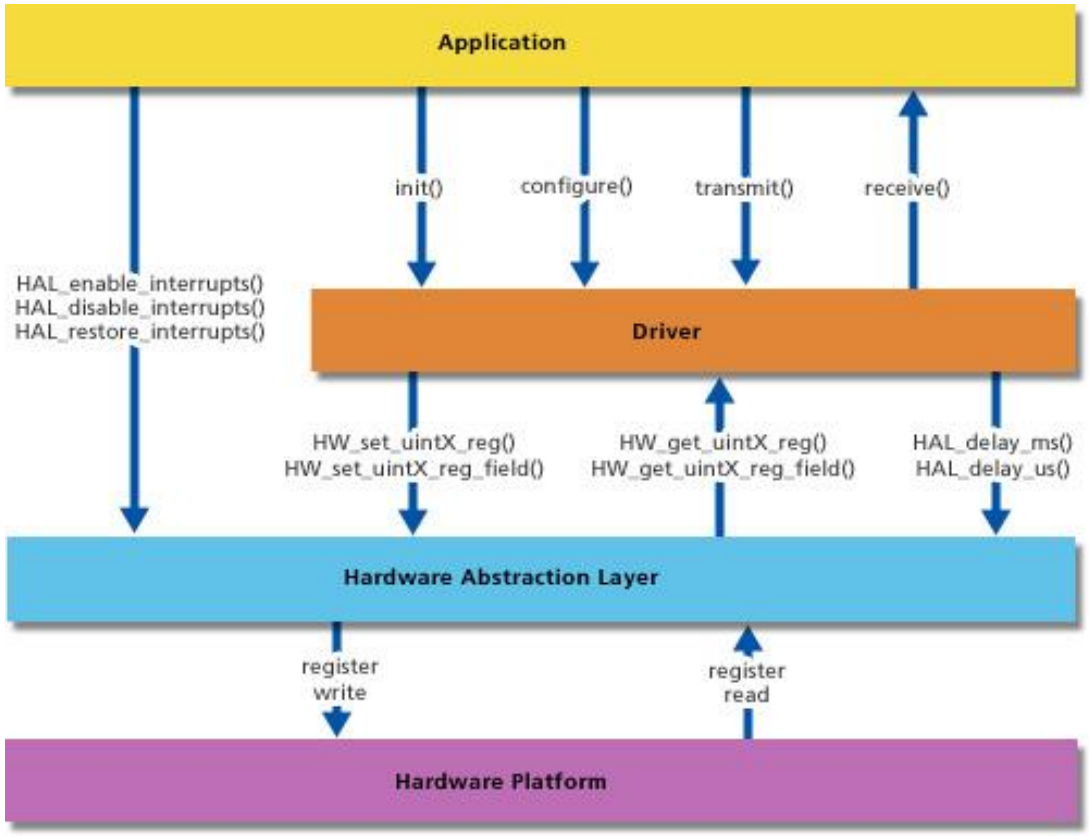
Hardware Abstraction Layers

- application is written using the RTOS API
- Real-Time Operating System (RTOS)
- Software Drivers: ease the use of the subsystem cores
- Hardware Abstraction Layers: used without modification with Cortex-M0, Cortex-M1



Hardware Abstraction Layers

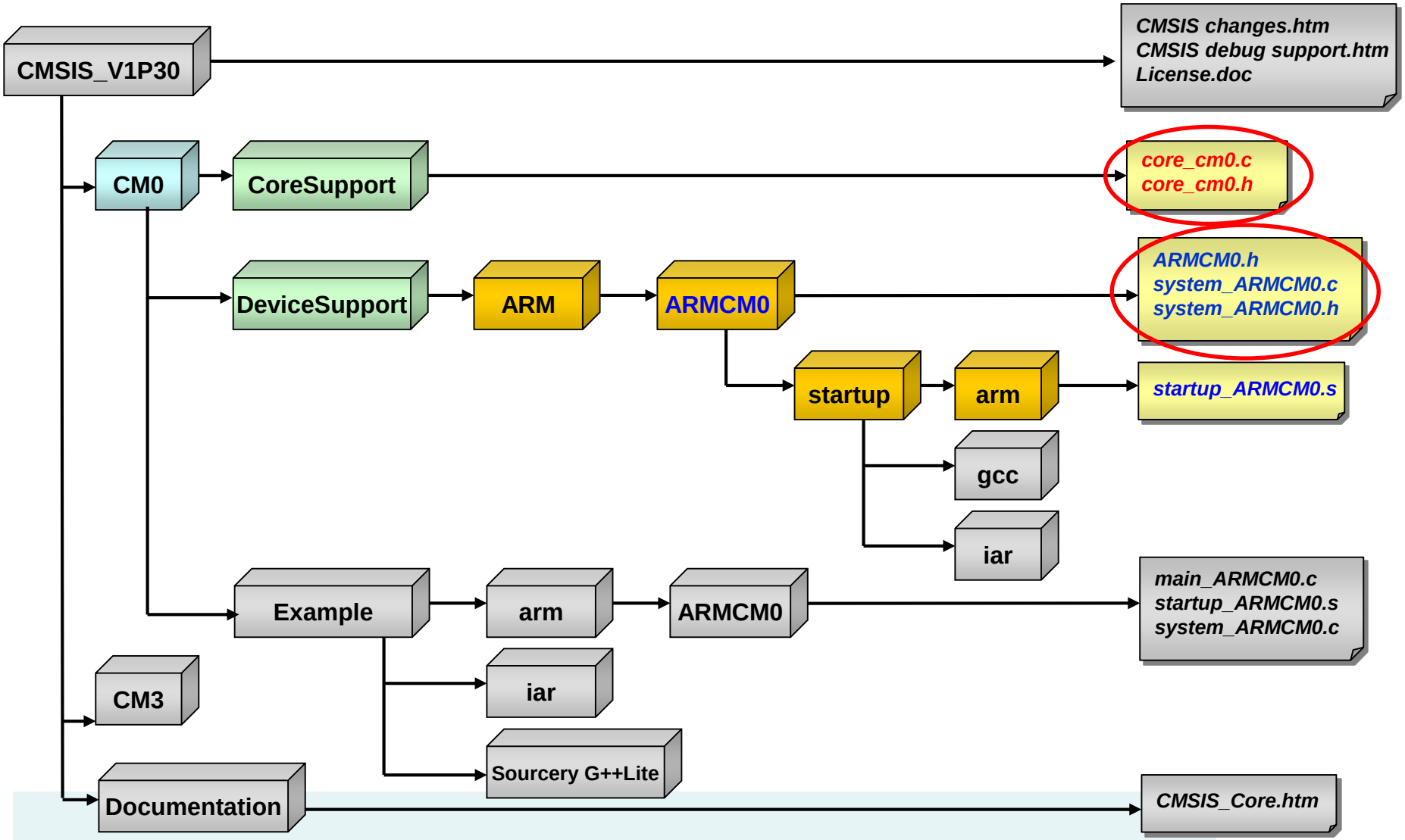
HAL translates the read and write requests relevant to hardware platform. reuse code even when the hardware platform changes.



CMSIS

- Cortex Microcontroller Software Interface Standard
- ARM& Tool vendor (Keil, IAR..)
- Structure
  - Core Peripheral Access Layer (CPAL)
  - Middleware Access Layer (MWAL) (ARM is current in development)
  - Device Peripheral Access Layer (DPAL)
    - Provide definitions for all device peripherals

CMSIS directory structure



CMSIS coding rule

- ANSI standard types defined in the ANSI C header file <stdint.h> are used
- For each exception/interrupt
  - An exception/interrupt handler rule
    - exception: the postfix _Handler
    - interrupts: the postfix _IRQHandler
  - A # define of the interrupt number with postfix _IRQn
  - A default exception/interrupt handler (**weak definition**) that contains an endless loop

What CMSIS Files Do You Need For NUC100 Series?

- NUC1xx.h(*device*.h)
  - Interrupt Number Definition
  - Device Peripheral Access Layer
    - Provides definitions for all device peripherals. It contains all data structures and the address mapping for the device specific peripherals
  - To define the peripherals for the actual device. It can use several other include files to define the peripherals of the actual devices
- startup_NUC1xx.s(startup_device.s)
  - Cortex-M0 startup code and the complete Interrupt Vector Table
- system_NUC1xx.c(system_device.c)

IC Vendor Offer 3files

- core_cm0.h
- core_cm0.c

ARM offer 2 files

NUC100 Interrupt Number Definition

```
typedef enum IRQn
{
/******  Cortex-M0 Processor Exceptions Numbers ***********************************************/
  NonMaskableInt_IRQn       = -14,    /*!< 2 Non Maskable Interrupt                */
  HardFault_IRQn            = -13,    /*!< 3 Cortex-M0 Hard Fault Interrupt        */
  SVCall_IRQn               = -5,     /*!< 11 Cortex-M0 SV Call Interrupt          */
  PendSV_IRQn               = -2,     /*!< 14 Cortex-M0 Pend SV Interrupt          */
  SysTick_IRQn              = -1,     /*!< 15 Cortex-M0 System Tick Interrupt      */
/******  ARMIKMCU Swift specific Interrupt Numbers ********************************************/
  BOD_IRQn                  = 0,
  WDT_IRQn                  = 1,
  EINT0_IRQn                = 2,
  EINT1_IRQn                = 3,
  GPAB_IRQn                 = 4,
  GPCDE_IRQn                = 5,
  PWM0_IRQn                 = 6,
  PWM1_IRQn                 = 7,
  TMR0_IRQn                 = 8,
  TMR1_IRQn                 = 9,
  TMR2_IRQn                 = 10,
  TMR3_IRQn                 = 11,
  UART0_IRQn                = 12,
  UART1_IRQn                = 13,
  SPI0_IRQn                 = 14,
  SPI1_IRQn                 = 15,
  SPI2_IRQn                 = 16,
  SPI3_IRQn                 = 17,
  I2C0_IRQn                 = 18,
  I2C1_IRQn                 = 19,
  CAN0_IRQn                 = 20,
  CAN1_IRQn                 = 21,
  SD_IRQn                   = 22,
  USBD_IRQn                 = 23,
  PS2_IRQn                  = 24,
  ACMP_IRQn                 = 25,
  PDMA_IRQn                 = 26,
  I2S_IRQn                  = 27,
  PWRWU_IRQn                = 28,
  ADC_IRQn                  = 29,
  DAC_IRQn                  = 30,
  RTC_IRQn                  = 31
} IRQn_Type;
```

The interrupt number is defined in the “NUC1xx.h” file

startup_NUC1xx.s (1/2)

```
_Vectors DCD    __initial_sp              ; Top of Stack
         DCD    Reset_Handler             ; Reset Handler
         DCD    NMI_Handler               ; NMI Handler
         DCD    HardFault_Handler         ; Hard Fault Handler
         DCD    0                         ; Reserved
         DCD    0                         ; Reserved
         DCD    0                         ; Reserved
         DCD    0                         ; Reserved
         DCD    0                         ; Reserved
         DCD    0                         ; Reserved
         DCD    0                         ; Reserved
         DCD    SVC_Handler               ; SVCall Handler
         DCD    0                         ; Reserved
         DCD    0                         ; Reserved
         DCD    PendSV_Handler            ; PendSV Handler
         DCD    SysTick_Handler           ; SysTick Handler
```

These exception names are fixed and define the start of the vector table for a Cortex-M0

startup_NUC1xx.s (2/2)

```
; External Interrupts
        DCD     BOD_IRQHandler
        DCD     WDT_IRQHandler
        DCD     EINT0_IRQHandler
        DCD     EINT1_IRQHandler
        DCD     GPAB_IRQHandler
        DCD     GPCDE_IRQHandler
        DCD     PWMA_IRQHandler
        DCD     PWMB_IRQHandler
        DCD     TMR0_IRQHandler
        DCD     TMR1_IRQHandler
        DCD     TMR2_IRQHandler
        DCD     TMR3_IRQHandler
        DCD     UART02_IRQHandler
        DCD     UART1_IRQHandler
        DCD     SPI0_IRQHandler
        DCD     SPI1_IRQHandler
        DCD     SPI2_IRQHandler
        DCD     SPI3_IRQHandler
        DCD     I2C0_IRQHandler
        DCD     I2C1_IRQHandler
        DCD     CAN0_IRQHandler
        DCD     Default_Handler
        DCD     Default_Handler
        DCD     USBD_IRQHandler
        DCD     PS2_IRQHandler
        DCD     ACMP_IRQHandler
        DCD     PDMA_IRQHandler
        DCD     I2S_IRQHandler
        DCD     PWRWU_IRQHandler
        DCD     ADC_IRQHandler
        DCD     Default_Handler
        DCD     RTC_IRQHandler
```

These interrupt names are fixed and define the start of the vector table for a Cortex-M0

PWRCON Register (Structure Bit Field SYSCLK_PWRCON_T Data Type)

31	30	29	28	27	26	25	24
RESERVED							
23	22	21	20	19	18	17	16
RESERVED							
15	14	13	12	11	10	9	8
							PD_WAIT_CPU
7	6	5	4	3	2	1	0
PWR_DOWN	PD_WU_STS	WINT_EN	WU_DLY	OSC10K_EN	OSC22M_EN	XTL32K_EN	XTL12M_EN

LSB

MSB

```
/*---------------------------- Clock Controller -----------------------------*/
typedef struct
{
    __IO uint32_t  XTL12M_EN:1;
    __IO uint32_t  XTL32K_EN:1;
    __IO uint32_t  OSC22M_EN:1;
    __IO uint32_t  OSC10K_EN:1;
    __IO uint32_t  WU_DLY:1;
    __IO uint32_t  WINT_EN:1;
    __IO uint32_t  INTSTS:1;
    __IO uint32_t  PWR_DOWN:1;
    __IO uint32_t  PD_WAIT_CPU:1;
    __I  uint32_t  RESERVE:23;
} SYSCLK_PWRCON_T;
```

New Data Type

Group Registers for fixed CLK_BA BaseAddr (SYSCLK_T Data Type)

```
typedef struct
{
    SYSCLK_PWRCON_T   PWRCON;
    SYSCLK_AHBCLK_T   AHBCLK;
    SYSCLK_APBCLK_T   APBCLK;
    uint32_t          RESERVED;
    SYSCLK_CLKSEL0_T  CLKSEL0;
    SYSCLK_CLKSEL1_T  CLKSEL1;
    SYSCLK_CLKDIV_T   CLKDIV;
    uint32_t          RESERVED2;
    SYSCLK_PLLCON_T   PLLCON;
    uint32_t          RESERVED3[3];
    SYSCLK_TREG_T     TREG;
} SYSCLK_T;
```

Register	Offset
PWRCON	CLK_BA + 00
AHBCLK	CLK_BA + 04
APBCLK	CLK_BA + 08
CLKSEL0	CLK_BA + 10
CLKSEL1	CLK_BA + 14
CLKSEL2	CLK_BA + 1C
CLKDIV	CLK_BA_+ 18
PLLCON	CLK_BA + 20
FRQDIV	GCR_BA + 24

To access register

```
#define AHB_BASE 0x50000000
#define SYSCLK_BASE    (AHB_BASE       + 0x00200)
#define SYSCLK          ((SYSCLK_T *) SYSCLK_BASE)
```

AHB Modules Space (0x5000_0000 – 0x501F_FFFF)		
0x5000_0000 – 0x5000_01FF	GCR_BA	System Global Control Registers
0x5000_0200 – 0x5000_02FF	CLK_BA	Clock Control Registers
0x5000_0300 – 0x5000_03FF	INT_BA	Interrupt Multiplexer Control Registers
0x5000_4000 – 0x5000_7FFF	GPIO_BA	GPIO Control Registers
0x5000_8000 – 0x5000_BFFF	PDMA_BA	SRAM_APB DMA Control Registers
0x5000_C000 – 0x5000_FFFF	FMC_BA	Flash Memory Control Registers

```
SYSCLK->PWRCON.XTL12M_EN = 1;
```

Cortex-M0 Core Registers Access (core_cm0.h)

Function Definition	Core Register	Description
void __enable_irq (void)	PRIMASK = 0	Global Interrupt enable (using the instruction **CPSIE i**)
void __disable_irq (void)	PRIMASK = 1	Global Interrupt disable (using the instruction **CPSID i**)
void __set_PRIMASK (uint32_t value)	PRIMASK = value	Assign value to Priority Mask Register (using the instruction **MSR**)
uint32_t __get_PRIMASK (void)	return PRIMASK	Return Priority Mask Register (using the instruction **MRS**)
void __set_CONTROL (uint32_t value)	CONTROL = value	Set CONTROL register value (using the instruction **MSR**)
uint32_t __get_CONTROL (void)	return CONTROL	Return Control Register Value (using the instruction **MRS**)
void __set_PSP (uint32_t TopOfProcStack)	PSP = TopOfProcStack	Set Process Stack Pointer value (using the instruction **MSR**)
uint32_t __get_PSP (void)	return PSP	Return Process Stack Pointer (using the instruction **MRS**)
void __set_MSP (uint32_t TopOfMainStack)	MSP = TopOfMainStack	Set Main Stack Pointer (using the instruction **MSR**)
uint32_t __get_MSP (void)	return MSP	Return Main Stack Pointer (using the instruction **MRS**)

Cortex-M0 Instruction Access

Function Name	CPU Instruction	Description
void __WFI (void)	WFI	Wait for Interrupt
void __WFE (void)	WFE	Wait for Event
void __SEV (void)	SEV	Set Event
void __ISB (void)	ISB	Instruction Synchronization Barrier
void __DSB (void)	DSB	Data Synchronization Barrier
void __DMB (void)	DMB	Data Memory Barrier
uint32_t __REV (uint32_t value)	REV	Reverse byte order in integer value.
uint32_t __REV16 (uint16_t value)	REV16	Reverse byte order in unsigned short value.
sint32_t __REVSH (sint16_t value)	REVSH	Reverse byte order in signed short value with sign extension to integer.

NVIC Setup Function Call

Function Name	Parameter	Description
void NVIC_SetPriorityGrouping (uint32_t PriorityGroup)	Priority Grouping Value	Set the Priority Grouping (Groups . Subgroups)
void NVIC_EnableIRQ (IRQn_Type IRQn)	IRQ Number	Enable IRQn
void NVIC_DisableIRQ (IRQn_Type IRQn)	IRQ Number	Disable IRQn
uint32_t NVIC_GetPendingIRQ (IRQn_Type IRQn)	IRQ Number	Return 1 if IRQn is pending else 0
void NVIC_SetPendingIRQ (IRQn_Type IRQn)	IRQ Number	Set IRQn Pending
void NVIC_ClearPendingIRQ (IRQn_Type IRQn)	IRQ Number	Clear IRQn Pending Status
void NVIC_SetPriority (IRQn_Type IRQn, uint32_t priority)	IRQ Number, Priority	Set Priority for IRQn
uint32_t NVIC_GetPriority (IRQn_Type IRQn)	IRQ Number	Get Priority for IRQn
uint32_t NVIC_EncodePriority (uint32_t PriorityGroup, uint32_t PreemptPriority, uint32_t SubPriority)	IRQ Number, Priority Group, Preemptive Priority, Sub Priority	Encode priority for given group, preemptive and sub priority
NVIC_DecodePriority (uint32_t Priority, uint32_t PriorityGroup, uint32_t* pPreemptPriority, uint32_t* pSubPriority)	IRQ Number, Priority, pointer to Priority Group, pointer to Preemptive Priority, pointer to Sub Priority	Decode given priority to group, preemptive and sub priority
void NVIC_SystemReset (void)	(void)	Resets the System

General Disclaimer

The Lecture is strictly used for educational purpose.

MAKES NO GUARANTEE OF VALIDITY

- **The lecture cannot guarantee the validity of the information found here.** The lecture may recently have been changed, vandalized or altered by someone whose opinion does not correspond with the state of knowledge in the relevant fields. Note that most other encyclopedias and reference works also have similar disclaimers.

No formal peer review

- The lecture is not uniformly peer reviewed; while readers may correct errors or engage in casual peer review, they have no legal duty to do so and thus all information read here is without any implied warranty of fitness for any purpose or use whatsoever. Even articles that have been vetted by informal peer review or *featured article* processes may later have been edited inappropriately, just before you view them.

No contract; limited license

- Please make sure that you understand that the information provided here is being provided freely, and that no kind of agreement or contract is created between you and the owners or users of this site, the owners of the servers upon which it is housed, the individual Wikipedia contributors, any project administrators, sysops or anyone else who is in *any way connected* with this project or sister projects subject to your claims against them directly. You are being granted a limited license to copy anything from this site; it does not create or imply any contractual or extracontractual liability on the part of Wikipedia or any of its agents, members, organizers or other users.
- There is **no agreement or understanding between you and the content provider** regarding your use or modification of this information beyond the Creative Commons Attribution-Sharealike 3.0 Unported License (CC-BY-SA) and the GNU Free Documentation License (GFDL);

General Disclaimer

Trademarks

- Any of the trademarks, service marks, collective marks, design rights or similar rights that are mentioned, used or cited in the lectures are the property of their respective owners. Their use here does not imply that you may use them for any purpose other than for the same or a similar informational use as contemplated by the original authors under the CC-BY-SA and GFDL licensing schemes. Unless otherwise stated , we are neither endorsed by nor affiliated with any of the holders of any such rights and as such we cannot grant any rights to use any otherwise protected materials. Your use of any such or similar incorporeal property is at your own risk.

Personality rights

- The lecture may portray an identifiable person who is alive or deceased recently. The use of images of living or recently deceased individuals is, in some jurisdictions, restricted by laws pertaining to personality rights, independent from their copyright status. Before using these types of content, please ensure that you have the right to use it under the laws which apply in the circumstances of your intended use. *You are solely responsible for ensuring that you do not infringe someone else's personality rights.*