STM8S Flash & Control System

May 2009

STM8S Technical Training

STM8S Memory main features (1/2)

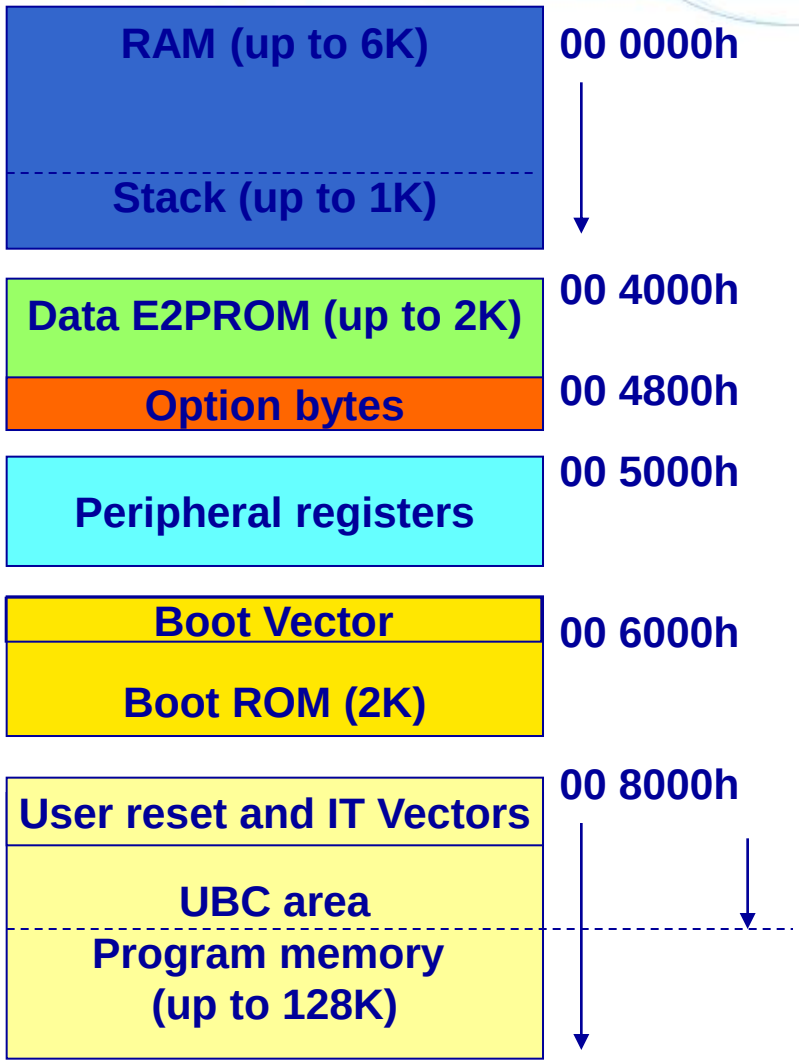
- Up to 128Kbyte Program memory (flash) based on EEPROM technology
- Up to 2K Data EEPROM memory
- Read While Write capability
- Write protection with Memory Access Security System (MASS keys)
- Programmable write protected User Boot Code area (UBC)
- Read Out Protection (ROP)

STM8S Memory main features (2/2)

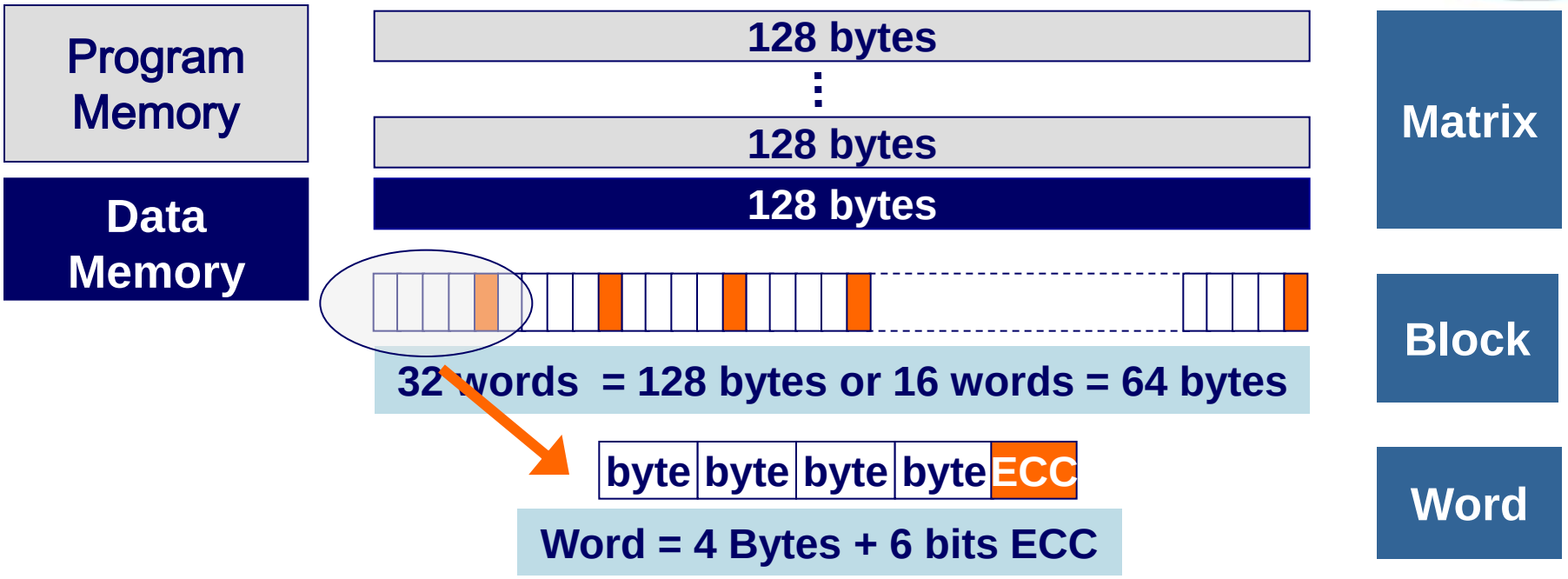
- Byte, word and block programming capability
- 6ms programming time (3ms erase + 3ms write)
- In Circuit Programming (ICP) and In Application Programming (IAP) capability
- One Interrupt vector dedicated to end of programming and attempt to write in protected area
- Configurable memory state in low power modes (Halt and Active Halt)
- Boot ROM embedding ST proprietary boot loader code

STM8S Family memory mapping

- The program memory always starts at address 00 8000h
  - End is determined by the memory size (02 7FFFh for 128K).
  - Interrupt vectors are located at the beginning of this memory area.
  - Programmable User Boot Code area includes the It vectors (from 1K up to 128K).
- The RAM starts from 00 0000h
  - Up to 1KBytes Stack is located at the end of the available RAM.
- The Data EEPROM starts from 00 4000h.
  - The option bytes are located after the available data EEP



Memory organization granularities



EEPROM access time allows to run the device up to 16 MHz at least without adding wait state.

For higher frequency (i.e. HSE@24Mhz), one wait state is necessary.

STM8S family memory ranges

- Low density
  - From 8 Kbytes of Flash Program ⇔ 128 pages of 64 bytes
  - 640 bytes of Data EEPROM ⇔ 10 pages of 64 bytes
- Medium density
  - From 16 to 32 Kbytes of Flash Program ⇔ 64 pages of 512 bytes
  - 1 Kbytes of Data EEPROM ⇔ 2 pages of 512 bytes
- High density
  - From 32 to 128 Kbytes of Flash Program ⇔ 256 pages of 512 bytes
  - Up to 2 Kbytes of Data EEPROM ⇔ 4 pages of 512 bytes

- For low density 1 page = 1 block
- For medium & high density 1 page = 4 blocks

Memory Access Security System (MASS) (Unlock mechanism)

- In order to prevent any unwanted write access to the Program memory, to the Data EEPROM and to the Option bytes, the whole NVM memory is **Write protected** by default **after a device reset**.
- The memory lock must be removed by software before any programming operation loading in the right order 2 MASS keys in a given Flash register.

	Unlock register	MASS Keys
Program Memory	FLASH_PUKR	0x56 (1°) 0xAE (2°)
Data Memory &Option	FLASH_DUKR	0xAE (1°) 0x56 (2°)

After programming it is safer to **re-protect** the given memory area by **clearing PUL or DUL bit** in the Flash status register **FLASH_IAPSR**.

Memory dedicated option bytes: Read-out protection

- There are two option bytes dedicated to memory protection: the **Read-out protection (ROP)** and the **UBC protection**

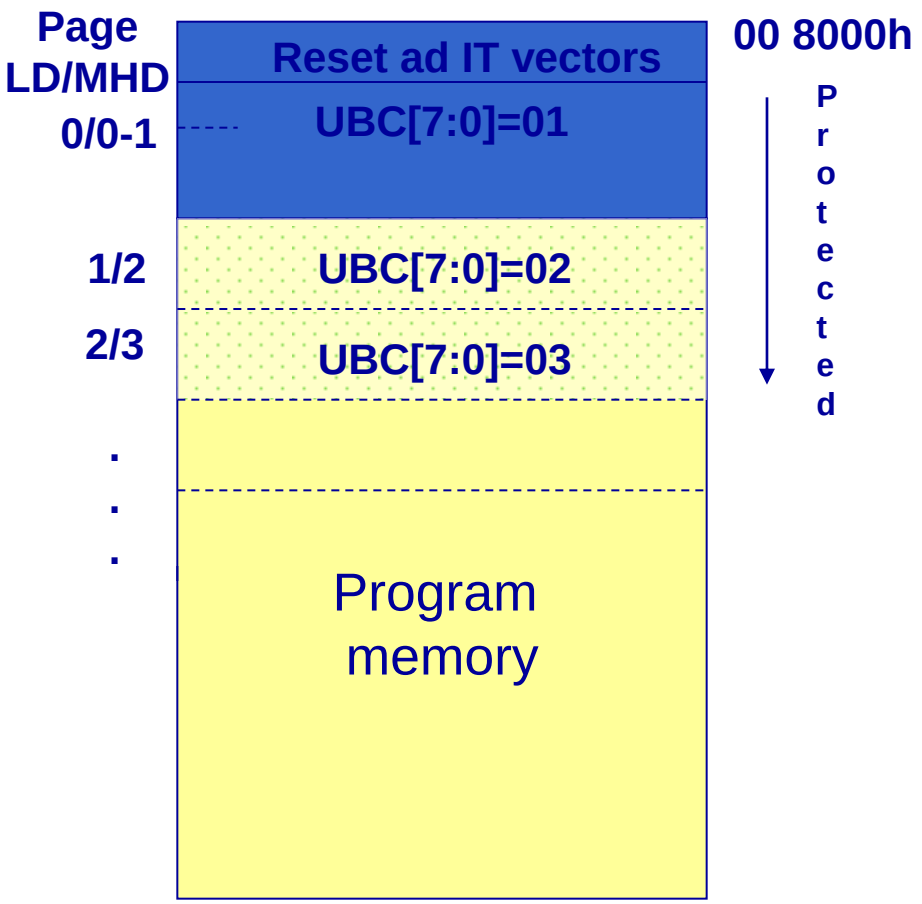
- When the **ROP is set**, it is **impossible to read back the memory**
  - Only embedded code can read the Program and Data memories
  - ICP/SWIM will not allow to read the memory, only the option byte can be read
  - Programming tools cannot read the memory

- It is possible to **remove the ROP in ICP/SWIM only**
  - But when it is removed, the **complete memory is erased**
  - **As a consequence we cannot proceed with Memory FAR**

ROP provides a reasonably good level of piracy protection.

Memory dedicated option bytes: UBC protection

- The goal of the **UBC protection** is to prevents a part of the memory from being written (especially in IAP), protecting critical application routine from unwanted modifications.
- When the UBC protection is active, the protected area always **include the reset and interrupts vectors**
- In order to activate the **protection**, **UBC[7:0] option byte** and its complement NUBC[7:0] must be set with the **number of pages** you want to protect: **this UBC area is not unlocked by the MASS keys.**
- To **unprotect** the **"UBC area"**, the option byte needs to be cleared (not available in IAP)



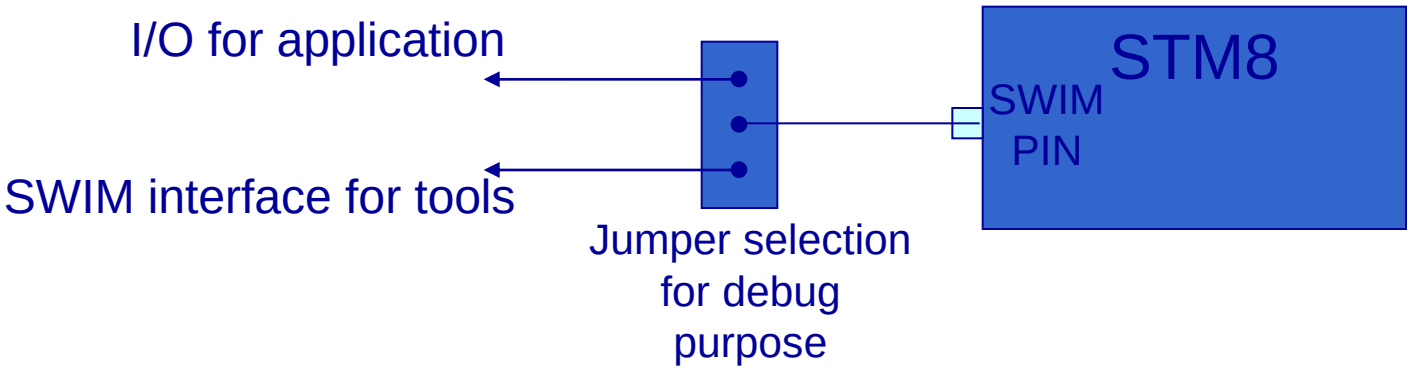
Memory programming methods

There are three methods to program the device memory:

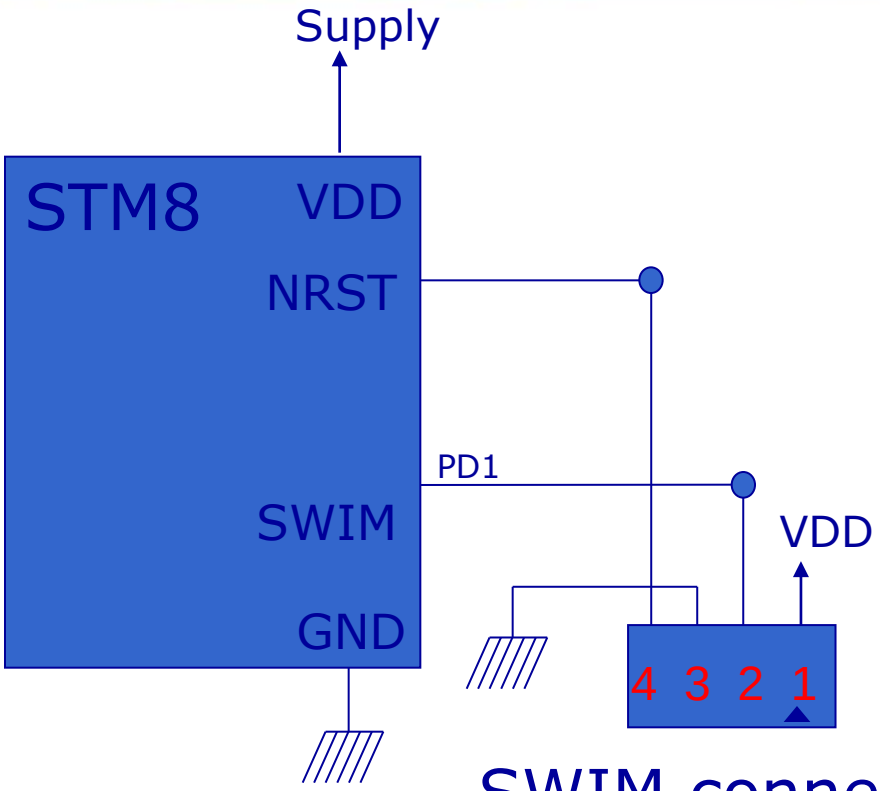
SWIM	Use a **programming tool with socket:** –The unsoldered part is placed in the programmer socket.
	Use **In Circuit Programming (ICP):** – The device is soldered on the application board – The programming tool is connected to the user application with the SWIM wires.
ST Boot Loader	Use the **Boot Loader:** – It allows the customer to store the application into the internal EEPROM using one of device serial interface (UART, LIN, CAN).
IAP	Use the **In Application Programming (IAP):** – The device is soldered on the application board and the application is running. – The application can reprogram itself receiving the new firmware through any available communication protocol. – The IAP code must have been programmed using one of the previous methods.

In Circuit Programming

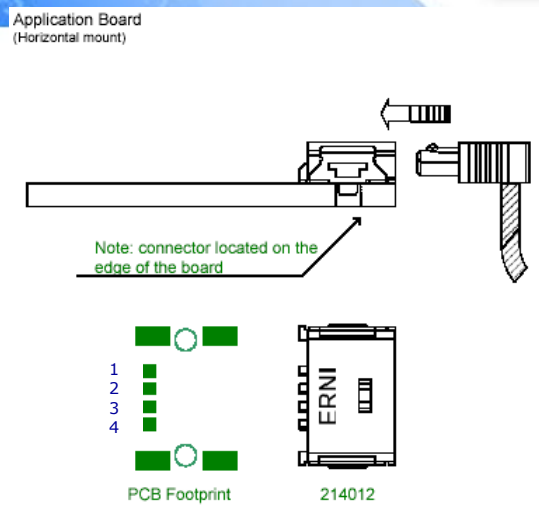
- ICP is using the SWIM (Single Wire Interface Module)
  - The SWIM protocol uses an hardware communication peripheral
  - It uses only one I/O that can also be used as PD1 GPIO pin.
    - By default the pin is in SWIM mode after reset
    - Switch to PD1 is done by software with a bit in the Global Configuration Register (CFG_GCR)
    - It is generally good to use a jumper on the application board.



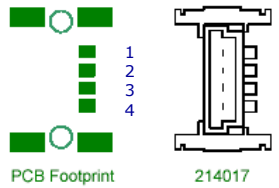
SWIM connection



SWIM connector



Dedicated connector with only 4-pins :
SWIM, Reset, GND, VDD (for monitoring)



For further information: http://www.erni.com

ST Boot Loader

- The **Boot Loader** is stored into the internal 2 Kbytes **Boot ROM** memory.

- This program uses **USART** (set as a normal UART), **CAN** or **LIN** peripherals (handled in polling mode) to **download/update** into Program memory, Data EEPROM and Option byte area respectively the **code** (including vector table), the **data** and the **option bytes**, thus using **ST proprietary protocol.**

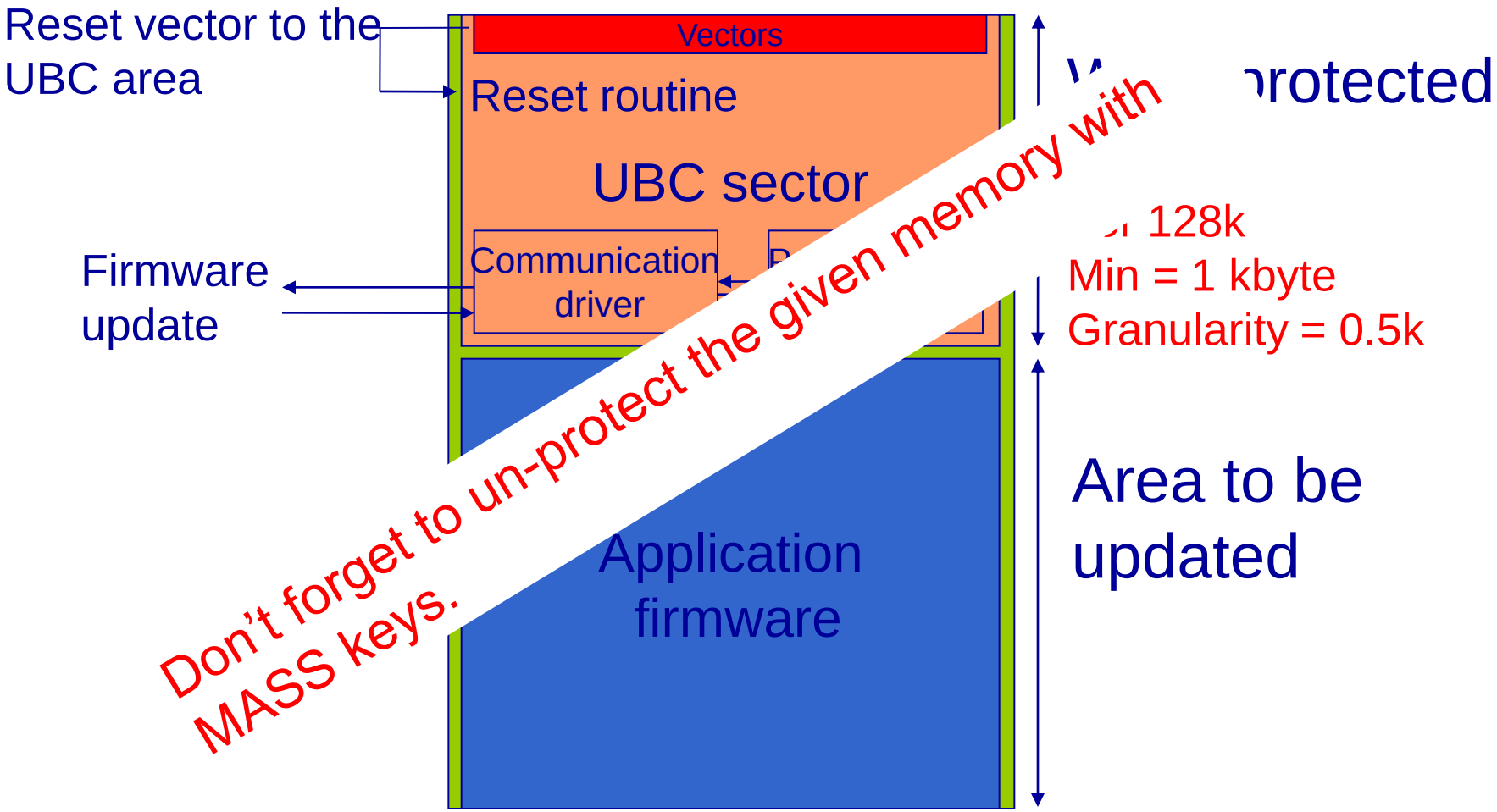
- According to the content of the Reset Vector location (0x8000) the host can decide to re-program or not the EEPROM by checking two option bytes, as showed in the following table:

	EEPROM Reset vector @0x8000	**Opt_byte @ 0x487E**	**Opt_byte @0x487F**	**Actual EEPROM status → *Action***
1°	XXh ≠(82h or ACh)	XXh	XXh	**EE virgin** → *jump to **BL***
2°	XXh=(82h or ACh)	55h	AAh	***EE already written*** → *jump to BL*
3°	XXh=(82h or ACh)	XXh ≠ 55h	XXh ≠ AAh	**EE already written** → *jump to EEPROM reset*

In Application Programming (1)

- In Application Programming
  - In case of in application programming, there is an **user boot sector** containing a **boot program** that will reprogram part of the application itself.
  - A new version of the firmware to be updated is received by the **boot program** using any possible protocol, usually a serial communication port (USART, SPI, I2C, CAN, LIN ….).
- The User Boot Code area feature answers to the IAP needs:
  - For the IAP to be safe, the boot sector must always be write protected.
  - In the event of a power failure, the boot program will be able to restart the firmware update from the beginning.
  - The boot sector contains the Reset vector which points into the boot program on the reset routine.
  - The boot program contains the update communication driver.

In Application Programming (2)



Byte programming

- Byte programming is done with a **simple load instruction** at the target address.
- If the write instruction is run **from Program memory** to itself, the **core is stalled** during the whole programming phase, otherwise if it is fetched from RAM the program can do something else (but no access to the program memory).
- The interrupt remain pending until the operation is not completed (core is stalled on IT when code fetched from RAM).
- If the write instruction is run **from Program memory to Data EEPROM** the **Read While Write (RWW) capability** allows to continue the program without constraints.
  - **RWW is NOT available for low density memory**
- Control flags can be used to check the end of programming status.
- The programming lasts **3ms** if the “word” was **empty** (no erase phase needed) or **6ms if not**.
- To erase a location, simply write 00h.

Don’t forget to unprotect the memory with MASS keys

Word programming

- Word programming allows the user to program 4 bytes with only one programming cycle which minimizes the programming time.
- The mechanism is similar to byte programming but the activation is done with dedicated bit and consecutive loads of 4 bytes in given word.
- The **Read While Write (RWW) capability** is available for medium and high density memories, **NOT for low density memory.**
- Control flags can be used to check the end of programming status.
- Word programming lasts always **6ms** (erase + write times).

Don’t forget to unprotect the memory with MASS keys

Block programming (1)

- The block programming feature is available for both the **Program memory** and the **Data EEPROM.**
- Block programming allows the user to program of **128 bytes** in one programming cycle; **the gain of time is very significant.**
- There are three block programming possible operations activated by **dedicated control bits:**
  - **Standard block programming:** the block is first erased then programmed.
  - **Fast block programming:** the block is not erased before programming. This mode has to be used only when the block is blank before operation; else the content is not guaranteed.
  - **Block erase:** The full block is erased. In this case it is enough to write '0' in the first word of the block (4 bytes).

Don't forget to unprotect the memory with MASS keys

Block programming (2)

- For block programming (Standard or fast block programming), the 128 data bytes have to be sequentially loaded. The actual programming phase starts automatically when the last byte is latched. **The initial address must be the first block address.**
- **Attention! The data loading operation for block writing has to be performed from the RAM even when addressing the Data EEPROM.**
- In case of **Program memory** programming after programming finishing only the soft can return executing in Program memory**.**
- In case of **Data EEPROM** programming the soft can return into Program memory as soon as dedicated flag HVOFF gives the green light when programming HV starts and thus to perform other tasks. It means that the **RWW capability** is available during the actual programming phase after the data loading.
- **The RWW is NOT available for low density memory**

Don’t forget to unprotect the memory with MASS keys

Option byte programming

- The option byte programming is very **similar to byte programming to Data EEPROM**.
- But there is an **additional control bit** to allow write access to option bytes.

- It is possible to **re-program the option bytes in IAP or user mode,** but the new option byte value is taken into account after next reset only.

Memory Programming

	Nber of Bytes	Programming from:	Programming to:	Core	Fast Programming	Programming Time
Byte	1	Program Mem	Program Mem	Stalled	Yes (Auto)	~3ms if the memory were empty or 6ms otherwise
		RAM	Program Mem	Running but stalled if interrupt occurs		
		RAM or Prog Mem	Data EEP or Opt Byte	Running No constraints: **RWW (1)**		
Word	4	Program Mem	Program Mem	Stalled	No	~6ms
		RAM	Program Mem	Running but stalled if interrupt occurs		
		RAM or Prog Mem	Data EEP or Opt Byte	Running No constraints: **RWW (1)**		
Block	128	Program Mem	Program Mem	**Not possible**	Yes (Ctrl Bit)	~3ms in Fast mode only if the memory were empty or 6ms in standard mode
		RAM	Program Mem	Running but stalled if interrupt occurs		
		RAM or Prog Mem Data loading from RAM ONLY	Data EEP or Opt Byte	Running No constraints: **RWW (1)**		

(1) In case of low density memory the RWW is not available. The core is stalled during programming from Program memory to Data EEP or Opt Bytes

References

- PM0051 STM8 Flash Programming Manual
- UM0560 STM8 Bootloader User Manual
- UM0462 STM32 and STM8 Flash Loader Demonstrator
- AN2659 STM8 in-application programming (IAP)

  using a customized user-bootloader

Quiz

- What is the use of IAP?
- How works the MASS?
- What is the specificity of the UBC?
- What is guaranteed thanks to the ROP?
- Which connection is needed on the application in order to do In Circuit Programming ?
- Is it possible to re-program option bytes in user mode?

STM8S CLOCK CONTROLLER

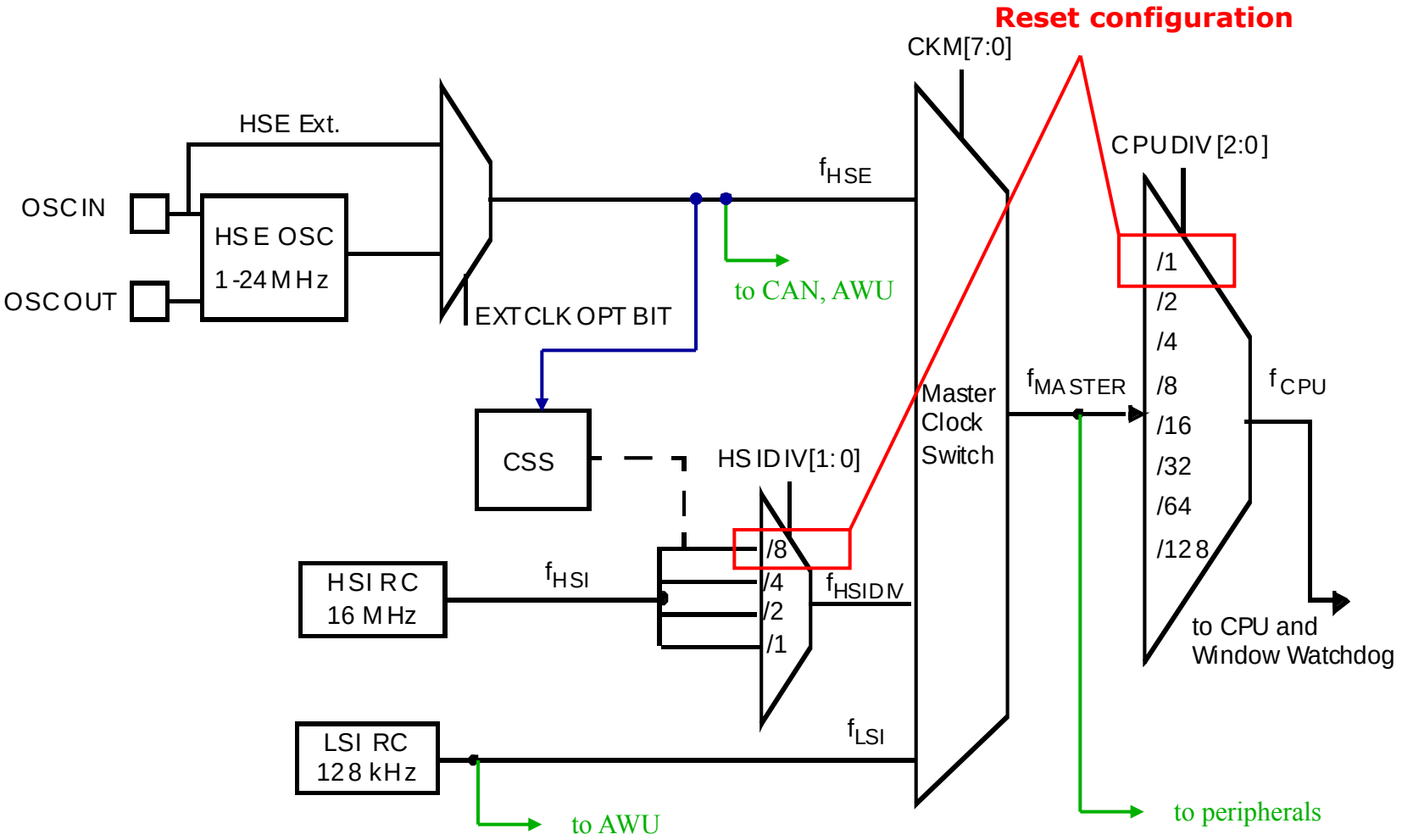
Features

- **4 clock sources:**
  - **HSE crystal**: 1-24 MHz **H**igh **S**peed **E**xternal crystal
  - **HSE user-ext**: Up to 24 MHz High Speed user-external clock
  - **HSI RC**: 16 MHz **H**igh **S**peed **I**nternal RC oscillator
  - **LSI RC**: 128 kHz **L**ow **S**peed **I**nternal RC

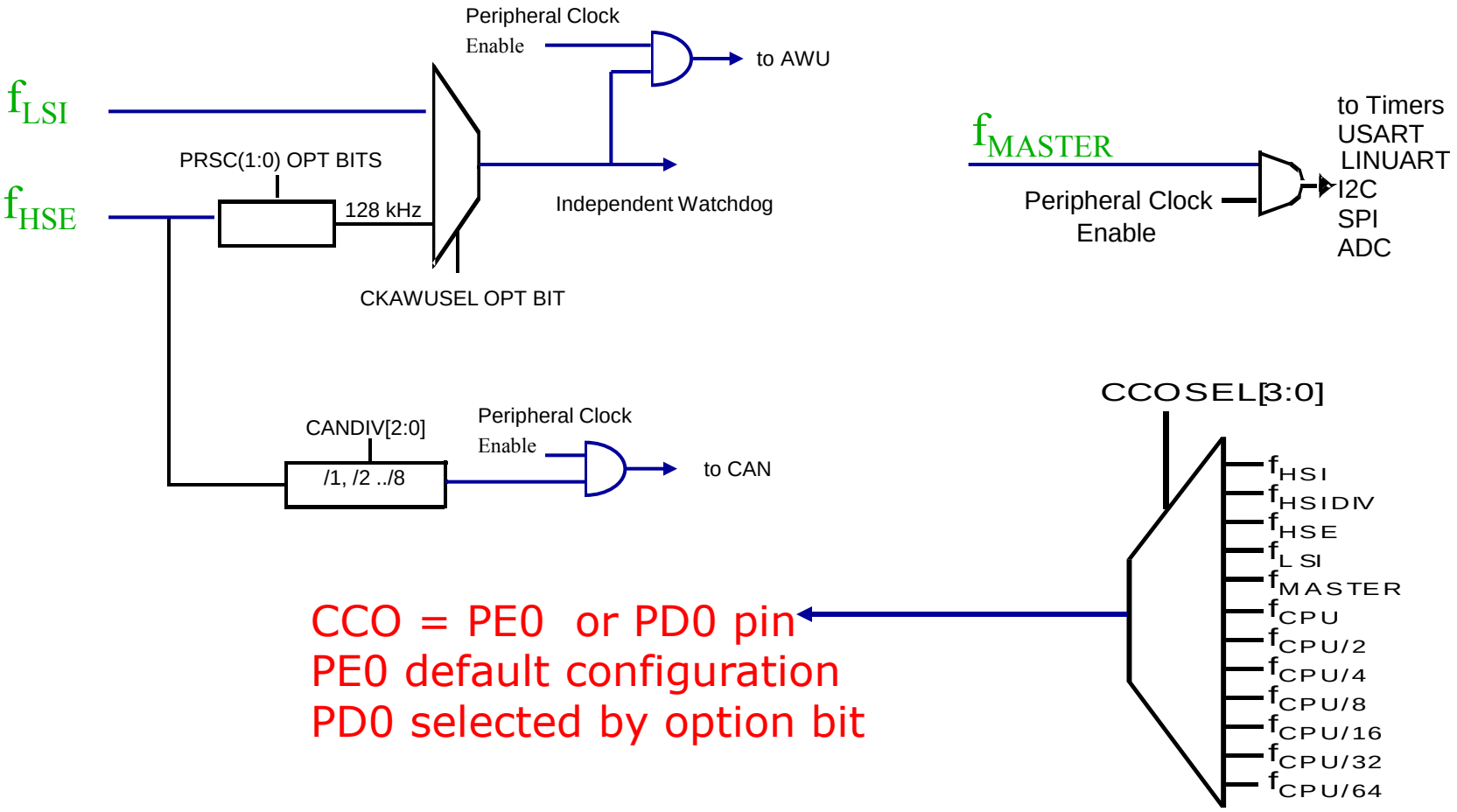
- **Efficient clock switching mechanism**
- **Clock Security System (CSS)**
- **Peripheral Clock Gating (PCG)**
- **Configurable Clock Output (CCO)**

- **EMS-hardened** clock configuration registers which values have to be complemented otherwise in case of corruption a reset is generated.

Block Diagram (1/2)



Block Diagram (1/2)



CCO = PE0 or PD0 pin
PE0 default configuration
PD0 selected by option bit

Clock Sources

- **All clock sources can be used as CPU clock source**
- **From 1 to 24MHz external crystal/ceramic resonator**
  - HSE OSCIN (PA1) and HSE OSCOUT (PA2)
  - From 3 V to 5.5 V supplied
  - -40 to 125 °C
  - 45-55 % duty cycle
  - < 2 ms max startup time @ 24 MHz
  - Bypass with HSE Ext. done by option bit
- **16MHz HSI RC factory trimmed**
  - < 2 µs max startup time
  - 1.8 V internally supplied, 1.2 µA max in Halt
  - ±1 % accuracy at 25 °C
  - ±5 % for the whole voltage and temp. range (3–5.5 V; -40 to 150 °C)
  - Further user trimming possible (8 steps to keep precision of 1.5 %)
- **128KHz LSI RC factory trimmed**
  - 1.8 V supplied, 0.3 µA max in Halt, ±2.5 % accuracy at 25 °C
  - ±12.5 % accuracy over full voltage and temp. range

Master Clock Switching

- The Clock Controller offers a fast and easy master clock (f_{MASTER}) switch feature with one dedicated interrupt (SWIE) and its associated flag (SWIF)

- There are 2 clock switch methods:
  - **Automatic**: switches to the clock source as soon as it is ready
    - SWEN=1, new clock master in CLK_SWR
  - **Manual**: allows to control the exact time of the clock switch
    - New clock in CLK_SWR, wait clock ready, SWEN=1

- After the switch, the previous clock has to be switched OFF manually if required for low power

CSS: Clock Security System

- Protection against HSE crystal clock source failure
- When a failure occurs:
  - Automatic connection of f_{MASTER} to the auxiliary clock source (HSI/8)
    - HSI prescaler can then be modified by the user if needed
  - HSE is switched off
- For safety reason, once enabled this CSS feature cannot be disabled anymore until next reset
- A dedicated interrupt associated to the CSS detection event and its flag are available (CSSDIE with CSSD flag)

PCG: Peripheral clock gating

- PCG allows to cut the clock feeding to selected peripherals in order to decrease the power consumption.

- Peripherals which can be gated:
  - Enabled by default
  - USART, LINUART, SPI, I2C, TIMERS, ADC
  - CAN, AWU (only clock system to register, not counter clock)

- To disable or enable the clock for each peripheral:
  - Reset/Set the corresponding PCKEN bit in CLK_PCKENR register
  - Then set the peripheral enable bit in the corresponding control register

CCO: Configurable Clock Output

- Steps to configure the required CCO:
  - Configure CCO output push pull (**set the related bit in CR1**)
  - Select clock to output through CLK_CCOR register
  - Enable CCO feature (CCOEN bit)

- The clock source selected in the CCO register will be automatically switched ON if not done previously by the user

- in order to know if the selected clock signal is output: check CCORDY bit

- If the CCO feature is switched OFF, the CCO pin has to be re-configured in input floating (default I/O state) by the user

Quiz

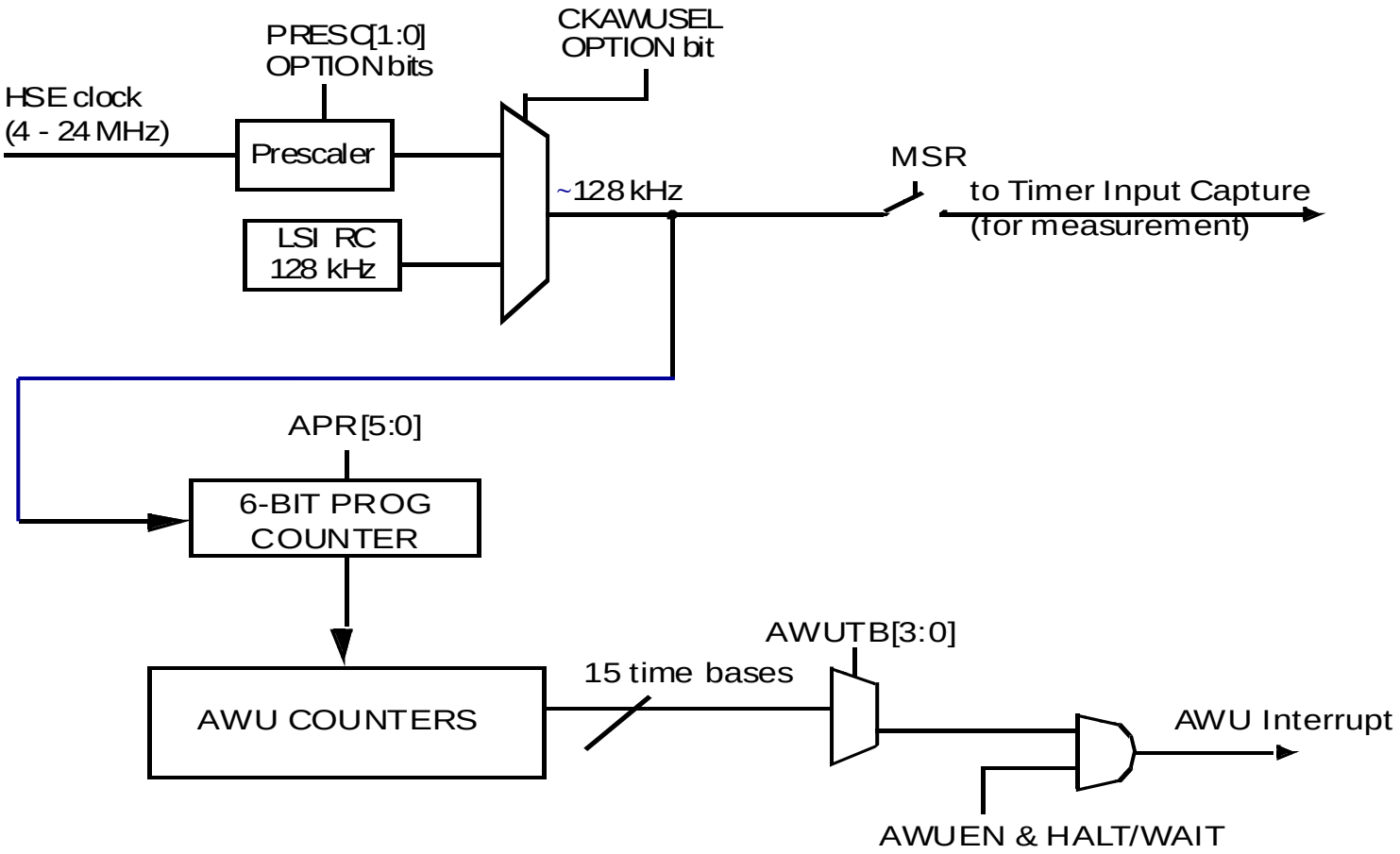
- What is the HSI RC value ?
- What is clock frequency at start-up?
- What are the master clock switching modes ?
- What is the auxiliary clock used by the CSS feature ?
- Can HSI be output on the CCO pin when Fmaster is using HSE ?
- Which are the two clock controller interrupts?

STM8S AUTO WAKE-UP (AWU) and BEEPER

Features

- **AWU function with Low Speed clock**
  - The AWU has been designed to implement the Active Halt low power mode.
  - Dedicated AWU interrupt vector
- **BEEPER function**
  - 1 kHz, 2 kHz or 4 kHz output signal frequencies
- **LSI frequency measurement**
  - It is possible thanks to LSI connection to one internal timer input capture. This operation allows to define precise time base for AWU and to make a calibration to keep Beeper frequency at 1K, 2K or 4K.

AWU Block Diagram



AWU description

- **AWU operation**
  1. LSI measurement if needed *(see dedicated slide)*
  2. Find the AWUTB[3:0] and APR[5:0] according to targeted wake-up interval *(see dedicated slide)*
  3. AWUEN=1 to enable the AWU interrupt and LS clock source to the counters (if LSI RC is selected it will start automatically)
  4. HALT
- **In Halt mode (after Halt instruction):**
  - the counters start to run when entering this mode
  - the AWU interrupt wakes-up the CPU at the regular programmed intervals :

This is the Active Halt mode

- **Power consumption reduction:** AWUTB[3:0] = “0000”
  - Puts AWU in stand-by: time-base OFF (default value)
- **APR[5:0] different from 3Fh (reset setting) to get counter running**

Time base selection (1/3)

- **The Time base selection is done with the determination of 2 parameters:**
  - APR[5:0], it gives the prescaler division factor APR_{DIV}
  - AWUTB[3:0], it gives the counter output rank
- **The chosen Time base depends on precise LSI frequency.**
- **We propose to define first 15 non overlapped ranges of intervals according to counter output rank.**
  - 0001 > $2/f_{LS}$ – $64/f_{LS}$, APR_{DIV} moving from 2 to 64
  - 0010 > $2x32/f_{LS}$ - $2x2x32/f_{LS}$, APR_{DIV} moving from 32 to 64 (std)
  - 0011 > $2x64/f_{LS}$ - $2x2x64/f_{LS}$, APR_{DIV} moving from 32 to 64 (std)
  - …
  - 1101 > $2^{11}x64/f_{LS}$ - $2^{11}x128/f_{LS}$, APR_{DIV} moving from 32 to 64 (std)
  - 1110 > $2^{11}x130/f_{LS}$ - $2^{11}x320/f_{LS}$, APR_{DIV} moving from 26 to 64
  - 1111 > $2^{11}x330/f_{LS}$ - $2^{12}x960/f_{LS}$, APR_{DIV} moving from 11 to 64
- **Important to measure LSI to get precise interval ranges**

Time base selection (2/3)

- **The targeted Time base is in one of the above ranges.**
  - It gives the TB value
- **Then the choice of APR_{DIV} is the result of one of the equations:**
  - Interval = $2^n x APR_{DIV}/f_{LS}$ where n depends on the TB value
  - Interval = $5x2^{11}xAPR_{DIV}/f_{LS}$
  - Interval = $30x2^{11}xAPR_{DIV}/f_{LS}$
- **The result is not necessarily an integer. It means that the targeted interval is reached with an error.**
- **The maximum error is half of the step in a given range; it is relatively more important for lowest targeted values.**
- **For standard range the maximum error is**
  - **0.5xstep/min range value -> 1/64 = 1.5625%**

Time base selection (3/3)

- **Example with f_{LS}=128KHz**
- **Target time interval is 3s**

The appropriate interval range is:

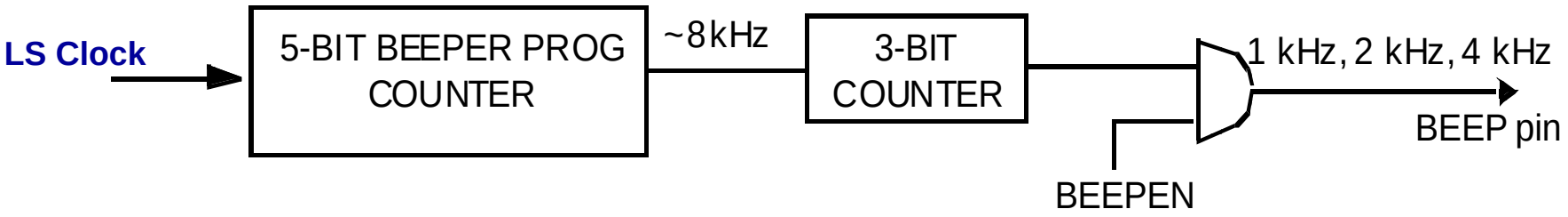
- 1110 : 2^{11}x130/f_{LS} - 2^{11}x320/f_{LS} = 2.08s – 5.12s

$3s = 5x2^{11}xAPR_{DIV}/f_{LS} \Rightarrow APR_{DIV} = 3xf_{LS}/5x2^{11} = 37.5$

The reached time interval is :

- $5x2^{11}x37/f_{LSE}$ = **2.96s** (or $5x2^{11}x38/f_{LSE}$ = **3.04s**)
- This is not precisely 3s
- Error = (3-2.96)/3 = 1.33%

Beeper Block Diagram



BEEP pin = PD4 when AFR7 option bit is set to 1

Beeper description

- **Beeper operation**
  1. LSI measurement and calibration if needed *(see dedicated slide)*
  2. Define BEEPDIV[4:0] values to get right $BEEP_{DIV}$ division factor
  3. Choice of BEEPSEL to select respectively: $f_{LS}/(8xBEEP_{DIV})$, $f_{LS}/(4xBEEP_{DIV})$ and $f_{LS}/(2xBEEP_{DIV})$
  4. BEEPEN =1

- **BEEPDIV[4:0] must be different from 1Fh (reset setting) to get counter running.**

LSI Calibration for Beeper

- **The LSI frequency is around 128KHz**
- **If you still need to adjust it to provide more accurate Beeper output at 1KHz, 2KHz or 4KHz the procedure below must be followed:**
- **Calibration procedure**
  1. MSR=1 to connect LSI to Timer TIM3 input capture 1
  2. Measure LSI frequency with TIM3
  3. Write the calibration value BEEPDIV[4:0] as follows:

     A and x being the integer and fractional part of $f_{LSI}/8$ in KHz the calibration value is:
     - BEEPDIV[4:0] = A-2 when x is less than or equal to A/(1+2*A)
     - BEEPDIV[4:0] = A-1 otherwise

Calibration (2/2)

- **Example with f_{LSI} = 115KHz:**

  $f_{LSI}/8$ = 14.375KHz
  -> A = 14
  -> A/(1+2*A) = 0.483

  -> x = 0.375
  -> x is less than A/(1+2*A) therefore:
  BEEPDIV[4:0] = A-2 = 12 = 0Ch

- The prescaler output frequency is
  - 115/14 = 8.2KHz
- Clock is calibrated for Beeper at 1.025KHz, 2.05KHz or 4.1KHz

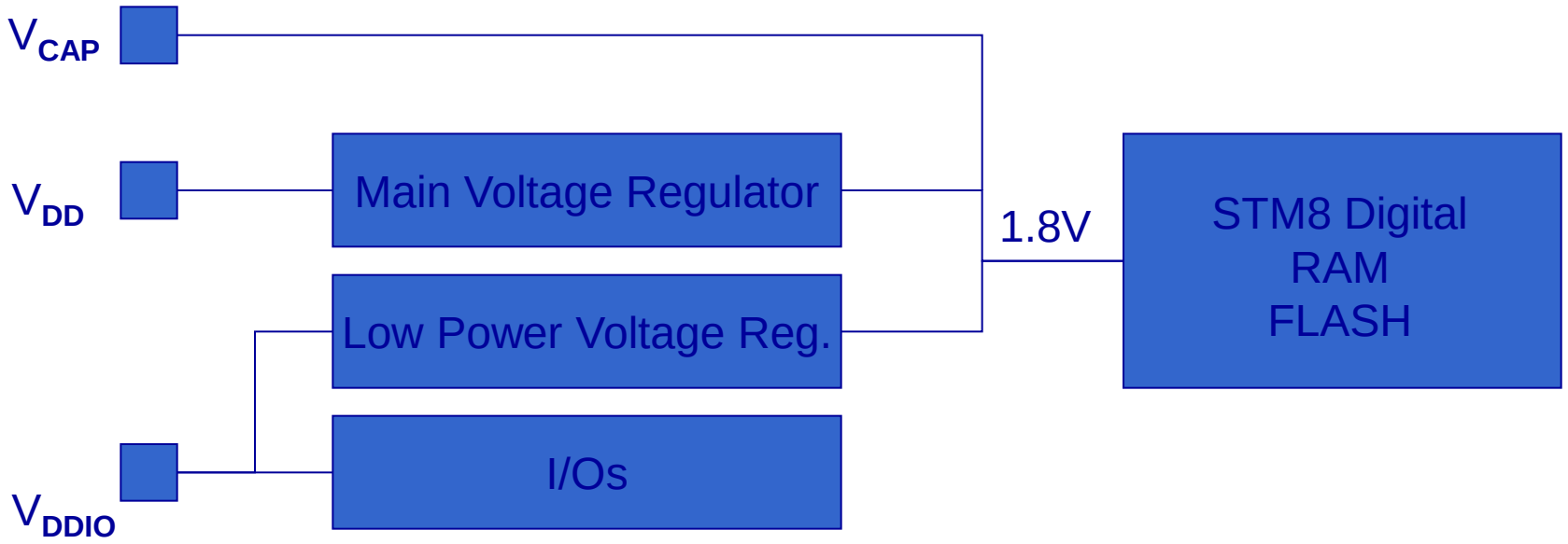
Quiz

- What are the clock sources used for AWU operations ?
  - Timer clock for LSI measurement
- What are the beeper output frequencies ?
- What is the BEEPDIV[4:0] correct value when LSI frequency is 128KHz ?

STM8S POWER SUPPLY, RESET AND VOLTAGE DETECTION

Power Supply Features

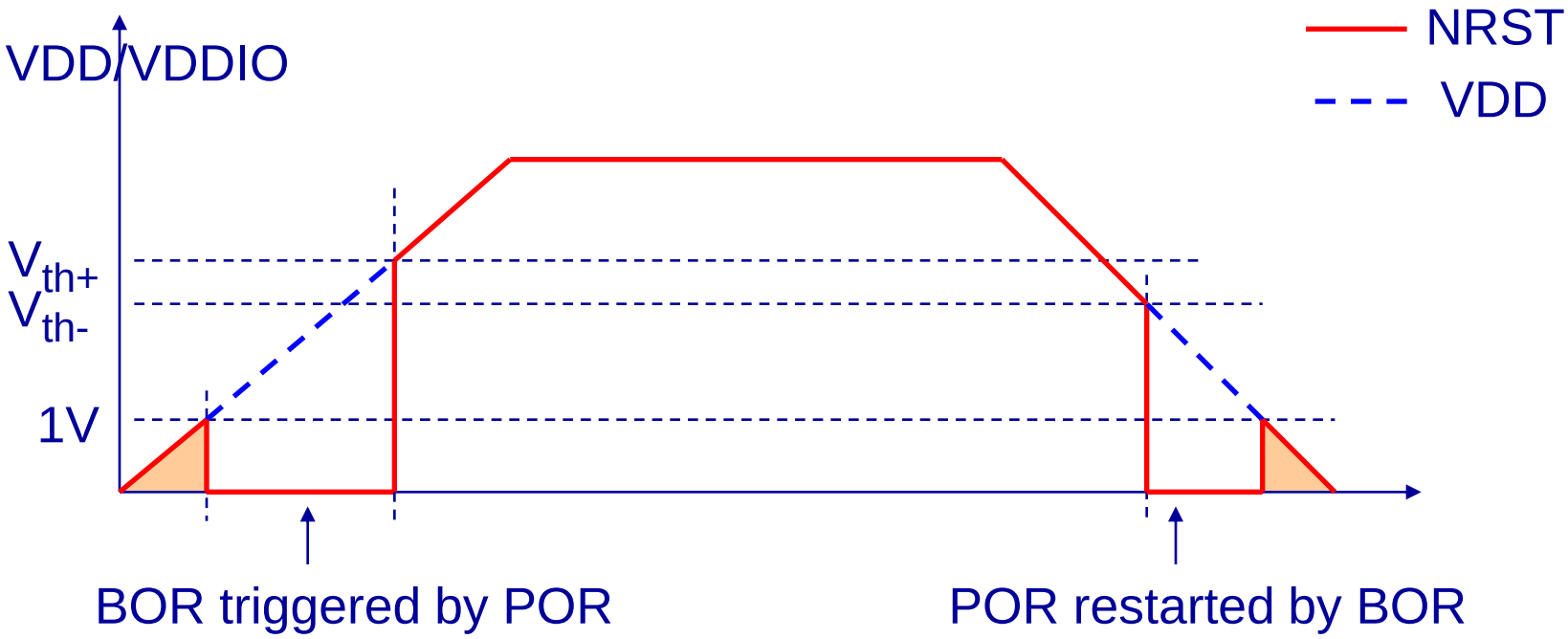
- There are 4 power supplies:
  - V_{DD}/V_{SS} - Main power supply
  - V_{DDIO}/V_{SSIO} - I/O ring power supply
  - V_{DDA}/V_{SSA} - Analog functions
  - V_{REF+}/V_{REF-} - ADC Reference



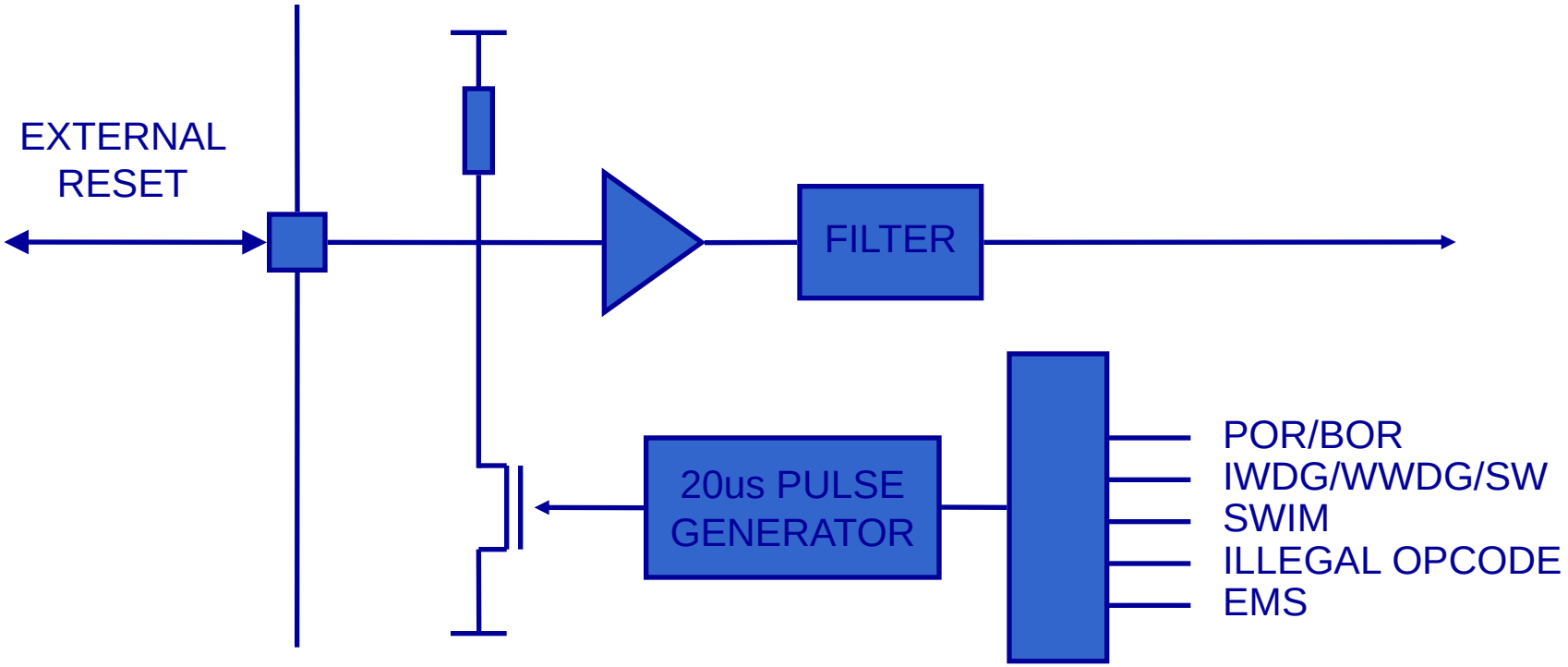
Refer to HW getting started Application Note

Voltage Detectors

- 2 Voltage Detectors
  - BOR to monitor main power supply
  - Power On Reset with Power Down Capability



Reset Circuit Block Diagram



External Reset

- **External reset pin: NRST**
  - In practice a negative **reset pulse** larger than **300ns** on NRST pin generates an internal reset system.
  - However NRST pin is also an **open-drain output** which delivers a **minimum pulse of 20µs** controlled by internal temporization.
  - Therefore the observed **reset pulse is 20µs minimum.**
  - There is an **internal weak pull-up** to maintain the reset pin at 1 when reset is not forced. It avoids spurious reset.

Internal Reset sources (1/2)

- **Power On Reset (POR):**
  - The POR is always active only during Power On (rising edge)
  - Generate trigger to start LVD, then switched off
- **BOR**
  - When voltage falls bellow level, generate trigger for POR
- **SWIM reset**
  - A specific SWIM command generates a reset when sent.
- **Illegal op-code reset**
  - In case of noise on the product corruption can lead to wrong op-code value which is detected and generates a reset.
- **Window Watchdog, Independent Watchdog and software resets.**
  - This 3 reset sources are directly controlled by software.

Internal Reset sources (2/2)

- **EMS reset**
  - The most critical registers of the product are implemented with their complement in such way any mismatch in the opposite values on EMS disturbances leads to a reset:
    - Copy of option byte registers,
      - UBC Protection registers for instance
    - Clock Controller switch registers,
      - Clock master register
      - Switch register
- **Watchdogs, Illegal op-code and EMS reset ensure product robustness allowing application to recover normal operation.**
- **Related flags can be tested and cleared in RST_SR**

STM8S versus ST7

- **ST7**
  - **LVD**
    - **+ Power-On Reset**
    - **+ Voltage Drop Reset**
    - **+ 3 thresholds**
    - **- Option byte switchable**
    - **- When deactivated, no more POR**
  - **AVD**
    - **Interrupt**
    - **3 thresholds**
    - **Activated with LVD**
    - **Option byte**
- **STM8**
  - **POR**
    - **Starts the system**
  - **BOR**
    - **Voltage Drop Reset**
    - **Fixed Threshold**
    - **Active during RUN**

Quiz

- What are the 8 internal reset sources?

.

- What are the reset sources implemented for robustness purpose?

- Which regulators are implemented?

STM8S POWER MANAGEMENT

Generalities (1/2)

- **In a silicon device there are two kind of consumption:**
  - **Static power consumption**
    - Due to analog polarization and leakage
    - It is negligible in RUN, but marginal for (Active) Halt
  - **Dynamic power consumption**
    - It depends on VDD, clock frequency and load capacitance
- **Microcontroller consumption depends on:**
  - MCU size (number of gates)
  - Design of analog features (linked to performances)
  - VDD
  - Clock frequency
  - Number of active peripherals (analog and digital)
  - Low power operating levels/modes

Generalities (2/2)

- **Low consumption feature is more and more important:**
  - Application on battery with very long life time
  - Minimization of the consumption contribution for global application system
  - Good for the environment

- **The challenge is to provide low power consumption keeping the system operating or able to reenter quickly in operation**.

- **Performances are very important to minimize the time the system is operating and therefore to maximize the time in inactive low power levels.**

Power management features

- **Flexible clock speed choices through:**
  - Clock sources
  - Various clock division choices
- **Peripheral Clock Gating**
- **Peripheral deactivation**
- **Several low power modes:**
  - Wait, Fast Active Halt, Slow Active Halt, Halt
  - Various consumption levels in each mode according to :
    - Clock speed and clock gating for Wait
    - Different voltage regulators for Fast/Slow Active Halt modes

Clock management

- **Slowing the system clocks:**
  - Choice of the clock sources HSE, HSI, LSI with programmable division factor for HSI
    - **Effective in both Run and Wait modes**
  - Choice of f_{MASTER} division factor to generate CPU clock
    - **Possible in Run mode only (so called Slow Mode)**
    - $f_{PERIPHERALS}$ remains on full speed
- **Peripheral clock gating:**
  - Give the possibility to switch-off by software the clock branch of non used peripherals
    - **Effective in both Run and Wait modes**
  - It doesn't mean the peripheral is off…
- **Each clock speed reduction or clock switch-off condition represent a different low power level.**

Peripheral switch-off

- **Each single peripheral can be deactivated to reduce the power consumption**
  - Except the watchdogs and the IO ports
- **It is true for both analog and digital peripherals**

Low power modes

- **The four Low power modes main characteristics:**

Modes / consumption	Oscillators	Main Regulator	CPU	Peripherals	Wake-up event
Wait -	ON	ON	OFF	ON (if no clk gating)	Int/ext ITs, BOR, reset
Fast Active Halt - -	OFF, except LSI/HSE	ON	OFF	AWU and IWDG if activated	AWU, ext ITs, BOR, reset
Slow Active Halt - - -	OFF, except LSI/HSE	OFF	OFF	AWU and IWDG if activated	AWU, ext ITs, BOR, reset
Halt - - - -	OFF	OFF	OFF	OFF	external ITs, BOR, reset

Low power modes (1/3)

- **Wait mode**
  - The MCU enters this mode when **Wait For Interrupt instruction** is executed.
  - CPU clock are stopped.
  - Registers and RAM contents are preserved.
  - The clock configuration defined before to enter in this mode remains unchanged.
  - The clock speed slow-down changing the clock source and PCG use are both means to reduce further the consumption.
  - Therefore there are again different low power levels for this low power Wait mode.

Low power modes (2/3)

- **Active Halt modes**
  - The MCU enters this mode when **Halt instruction** is executed.
    - **And if AWU and IWD are activated before.**
  - All clocks are stopped; the CPU and all the peripherals are disable by definition.
    - **Except LSI or HSE divided**
  - Digital part of MCU consumes almost no power but leakages.
    - **Except running AWU and IWD counters**
  - Registers and RAM contents are preserved.
  - By default the clock configuration defined before to enter in this mode remains unchanged.
- **Slow Active Halt versus Fast Active Halt:**
  - **When “Slow Active Halt” bit = “1”, Low Power Regulator is used instead of Main Power Regulator**
  - **On the contrary, Main Voltage Regulator remains active in Fast Active Halt to ensure fast startup (“Slow Active Halt” bit = “0”)**

Low power modes (3/3)

- **Halt mode**
  - The MCU enters this mode when **Halt instruction** is executed.
    - **And if IWD and AWU are not activated**
  - All clocks are stopped; the CPU and all the peripherals are disabled by definition; Main Regulator is switched off
  - Digital part of MCU consumes almost no power but leakages.
  - Registers and RAM contents are preserved.
  - By default the clock configuration defined before to enter in this mode remains unchanged.

Additional power controls (1/3)

- **Fast clock wake-up feature:**
  - In the clock controller there is the possibility thanks to the **FHW bit** in the Internal Clock Register (CLK_ICKR) to force the HSI clock when MCU returns from Halt or Active Halt.
  - HSI clock because the HSI RC oscillator start-up time is very short compare to HSE Crystal oscillator
  - **This possibility is important to optimize the time during which the MCU is active between two low power periods. It is in complement of pure CPU performance.**

- **IRET behaviour configuration (CFG_GCR.AL)**
  - Retrieve context and return to main (default, AL=0)
  - **OR return back to WFI/HALT** (AL=1)
    - Context saved on first IT only
    - Context restored when AL is reset in IT before IRET

Additional power controls (2/3)

- **Fast Flash wake-up from Halt mode**
  - In Halt mode, by default the Flash is in **Power-down mode** with negligible current leakages to ensure a very low Halt consumption.
  - But the Flash wake-up time is long.
  - However if the **HALT bit** in Flash control register 1 (FLASH_CR1) is set the Flash switches to **Operating mode** when MCU enters Halt mode.
    - **It ensures a fast wake-up time.**
  - **But the Halt consumption increases up to a few µAmps.**

Additional power controls (3/3)

- **Very low Flash consumption in Active Halt mode**
  - In Active Halt mode, by default the Flash remains in **Operating mode** to ensure the best wake-up time.
  - But the Flash consumption is not negligible around a few µAmps.
  - However if the **AHALT bit** in Flash control register 1 (FLASH_CR1) is set the Flash switches to **Power-down mode** when MCU enters Active Halt mode.
    - **It ensures a negligible Flash consumption.**
  - **But the Active Halt wake-up time increases.**

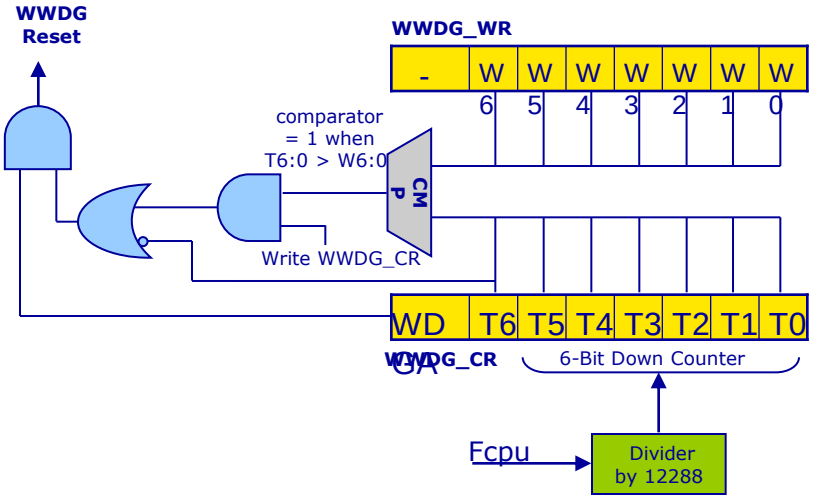
Quiz

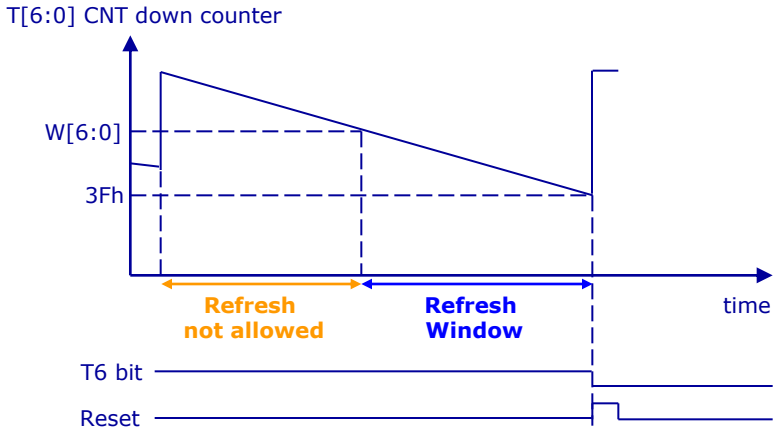
- What are the low power modes?
- How many low power levels are available?
- What are the five main parameters influencing silicon device power consumption?
- What are the possible trigger events to wake-up from Active Halt?

Window Watchdog (WWDG)

WWDG features

- Window Watchdog can be activated by Option byte or by software (setting WDGA)
- Configurable time-window, can be programmed to detect abnormally early or late application behavior
- Conditional reset
  - Reset when the down counter value becomes less than 40h (T6=0)
  - Reset if the down counter is reloaded outside the time-window
- To prevent WWDG reset: write T[6:0] bits at regular intervals while the counter value is lower than the time-window value (W[6:0])
- WWDG reset flag (in RST_STAT) to inform when a WWDG reset has occured
- Min-max timeout value @16MHz : 0.768ms/49.152ms by step of 0.768ms



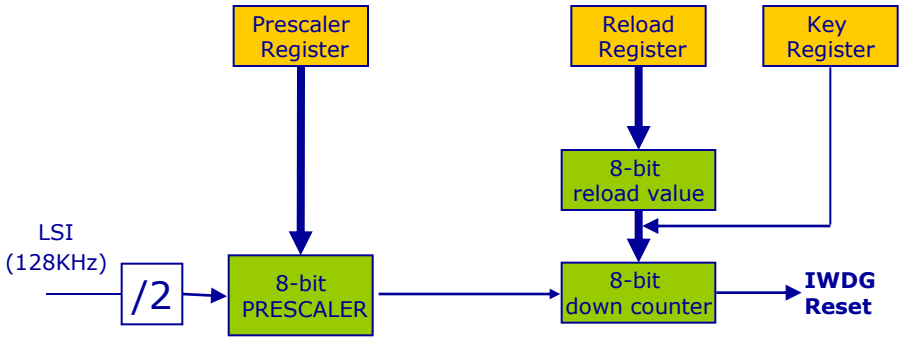


Best suited to applications which require the watchdog to react within an accurate timing window

Independent Watchdog (IWDG)

IWDG features

- Selectable HW/SW start through option byte
- Advanced security features:
  - IWDG clocked by its own dedicated low-speed clock (LSI) and thus stays active even if the main clock fails
  - Once enabled the IWDG can't be disabled (LSI can't be disabled too)
  - Safe Reload Sequence (by key)
  - IWDG is still functional all low power modes except in halt (LSI stopped)
- To prevent IWDG reset: write IWDG_KR with AAh key value at regular intervals before the counter reaches 0
- IWDG reset flag (in RST_STAT) to inform when a IWDG reset occurs
- Min-max timeout value @64kHz (LSI/2): 62.5us / 1s



Best suited to applications which require the watchdog to run as a totally independent process outside the main application

Quiz

- How the WWDG reset is refreshed ?
- When must be refreshed the WWDG reset ?
- Which clock feeds the IWDG down counter ?
- How is refreshed the IWDG ?
- What is the maximum IWDG timeout ?

Reference

- RM0016 STM8S Reference Manual
- AN2752 Getting Start with STM8
- AN2780 Real-time keeping on STM8S devices and usage of AWU and beeper
- AN2822 STM8S HSI Oscillator Calibration
- AN2857 STM8S Family Power Management

CONFIDENTIALITY OBLIGATIONS:

THIS DOCUMENT CONTAINS SENSITIVE INFORMATION.

IT IS CLASSIFIED “MICROCONTROLLERS, MEMORIES & SECURE MCUs (MMS) RESTRICTED AND ITS DISTRIBUTION IS SUBMITTED TO ST/MMS AUTHORIZATION

AT ALL TIMES YOU SHOULD COMPLY WITH THE FOLLOWING SECURITY RULES:

- DO NOT COPY OR REPRODUCE ALL OR PART OF THIS DOCUMENT
- KEEP THIS DOCUMENT LOCKED AWAY
- FURTHER COPIES CAN BE PROVIDED ON A “NEED TO KNOW BASIS”, PLEASE CONTACT YOUR LOCAL ST SALES OFFICE OR DOCUMENT WRITTER.