Adaptation and Evaluation of a Post-Quantum Cryptographic Algorithm on a Resource-Constrained Device

Case Study: SQIsign on ARM Cortex-M4 Microcontroller

Main « Actors »

- SQIsign
- GMP
- ARM Cortex M4
- STM32

SQIsign

- Digital signature scheme (Based on elliptic curve, isogeny and quaternion)
- Participate NIST competition
- Need to give example for level security 1,3 and 5

Levels of security required by NIST

1) Any attack that breaks the relevant security definition must require computational resources comparable to or greater than those required for key search on a block cipher with a 128-bit key (e.g., AES-128)
2) Any attack that breaks the relevant security definition must require computational resources comparable to or greater than those required for collision search on a 256-bit hash function (e.g., SHA-256/ SHA3-256)
3) Any attack that breaks the relevant security definition must require computational resources comparable to or greater than those required for key search on a block cipher with a 192-bit key (e.g., AES-192)
4) Any attack that breaks the relevant security definition must require computational resources comparable to or greater than those required for collision search on a 384-bit hash function (e.g., SHA-384/ SHA3-384)
5) Any attack that breaks the relevant security definition must require computational resources comparable to or greater than those required for key search on a block cipher with a 256-bit key (e.g., AES-256)

Source : https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/call-for-proposals-dig-sig-sept-2022.pdf

GMP

- “GNU Multiple Precision Arithmetic Library”
  - Use for multiple precision to compute
  - Use for the calculation inside Quaternions part

ARM Cortex M4

- TODO

STM32

- TODO

State of the Art

The verification routines of the reference implementation are compatible with 32-bit embedded systems based on e.g., the ARM Cortex-M4 core. However, the build system and provided applications are not suitable for a bare-metal environment as typically provided in such systems. The recommended platform for evaluating SQISIGN on the ARM Cortex-M4 core is the `pqm4` project [KPR+]. A helper shell script, to copy the source files to a folder structure compatible with `pqm4`, is provided in the submission package; instructions for its use can be found in the `README.md` file.

Source : https://sqisign.org/spec/sqisign-20250205.pdf

State of the Art

The verification routines of the reference implementation are compatible with 32-bit embedded systems based on e.g., the ARM Cortex-M4 core. However, the build system and provided applications are not suitable for a bare-metal environment as typically provided in such systems. The recommended platform for evaluating SQISIGN on the ARM Cortex-M4 core is the `pqm4` project [KPR+]. A helper shell script, to copy the source files to a folder structure compatible with `pqm4`, is provided in the submission package; instructions for its use can be found in the `README.md` file.

Source : https://sqisign.org/spec/sqisign-20250205.pdf

State of the Art

The verification routines of the reference implementation are compatible with 32-bit embedded systems based on e.g., the ARM Cortex-M4 core. However, the build system and provided applications are not suitable for a bare-metal environment as typically provided in such systems. The recommended platform for evaluating SQISIGN on the ARM Cortex-M4 core is the `pqm4` project [KPR+]. A helper shell script, to copy the source files to a folder structure compatible with `pqm4`, is provided in the submission package; instructions for its use can be found in the `README.md` file.

Source : https://sqisign.org/spec/sqisign-20250205.pdf

Experimentation

- Phase 1 : Adaptation
- Phase 2 : Evaluate SQIsign
- Phase 3 : Implementation of a solution
- Phase 4 : The Bug
- Phase 5 : Evaluate SQIsign with all level of difficulties
- Phase 6 : Data gathering

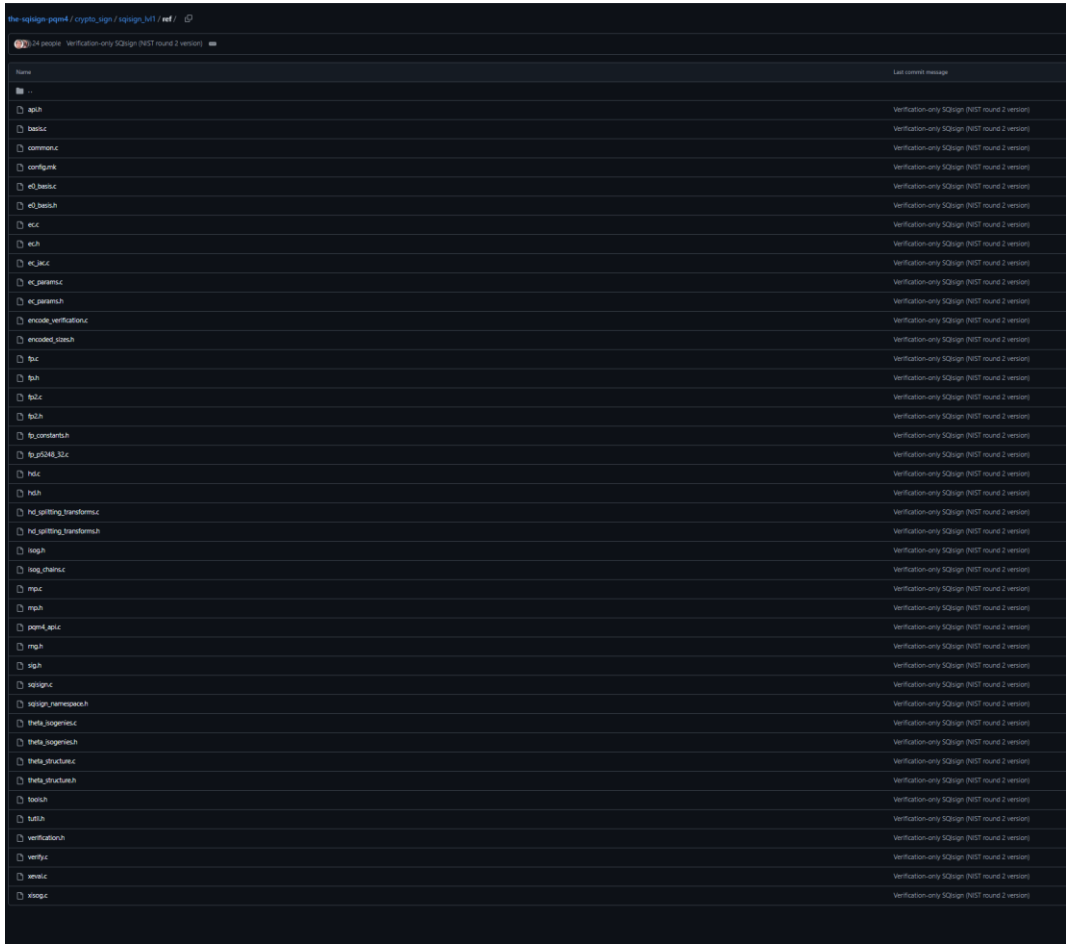
Phase 1

- Update build system for bare-metal environment
- Update applications for bare-metal environment

SQIsign (version 2) : all files

Update build system for bare-metal environment

- Example of the works for verification
  - All files has been flattened
  - No more Cmake



Source : https://github.com/SQISign/the-sqisign-pqm4/tree/SQIsign/crypto_sign/sqisign_lvl1/ref

Update build system for bare-metal environment

- Main Goal of the SQIsign team is to adapt their code for PQM4
  - PQM4 :
    - Collection of post-quantum cryptographic algorithms for the ARM Cortex-M4
    - benchmarking and testing framework

- My Goal is to adapt to STM32CubeIDE

=> The same steps

What are the implications of not using current build system ?

- No more automatization in term of variable selection
  - Manually add to the code
- Need duplicate the work by 3 (for level security 1,3 and 5) select files dependent of the level security
  - Manual selection
- Need to select file manually between 32 bits and 64 bits architecture
- Need to be carefull to not select file optimize for Broadwell
- …

Exemple_nistapi.c

- Basic file of SQIsign project which contain :
  - Key Generation
  - Signature
  - Verification
  - Small tests to evaluate each result

Update build system for bare-metal environment

- Methodology
  - Begin with an empty makefile
  - Use exemple_nistapi.c as base file
    - Try to compile
    - Get an error
    - Solve it

Case 1

- Undifined function => grep and find the function => add file to makefile
  - Main issue => “sqisign_namespace.h”

```
#define PARAM_JOIN3_(a, b, c) sqisign_##a##_##b##_##c
#define PARAM_JOIN3(a, b, c) PARAM_JOIN3_(a, b, c)
#define PARAM_NAME3(end, s) PARAM_JOIN3(SQISIGN_VARIANT, end, s)
```

Rename function

Grep does not work anymore

Case 2

- Selection of variable :
  - Reverse the code to make a choice
  - Read “Algorithm specifications and supporting documentation”

```
#if RADIX == 32
#if defined(SQISIGN_GF_IMPL_BROADWELL)
#define NWORDS_FIELD 8
#else
#define NWORDS_FIELD 9
#endif
#define NWORDS_ORDER 8
#elif RADIX == 64
#if defined(SQISIGN_GF_IMPL_BROADWELL)
#define NWORDS_FIELD 4
#else
#define NWORDS_FIELD 5
#endif
#define NWORDS_ORDER 4
#endif
#define BITS 256
#define LOG2P 8
```

Exemple inside the file fp_constants.h

```
#if defined(ENABLE_SIGN)
SQISIGN_API
int
crypto_sign_keypair(unsigned char *pk, unsigned char *sk);

SQISIGN_API
int
crypto_sign(unsigned char *sm, unsigned long long *smlen,
            const unsigned char *m, unsigned long long mlen,
            const unsigned char *sk);
#endif

SQISIGN_API
int
crypto_sign_open(unsigned char *m, unsigned long long *mlen,
                 const unsigned char *sm, unsigned long long smlen,
                 const unsigned char *pk);
```

Exemple inside the file api.h

Case 2

```
#if defined(STM32)
  #include "macro_m4.h"
#endif
```

```
#define MINI_GMP
#define RADIX_32
#define GMP_LIMB_BITS 32
#define RADIX 32
#define RANDOMBYTES_SYSTEM
#define ENABLE_SIGN

// #define NDEBUG
// #define DEBUG_VERBOSE
// #define MINI_GMP_DONT_USE_FLOAT_H
```

Case 3

- Change #include <file> => #include "file" when a file is related to the project SQIsign
  - STM32 need that for /Core/Inc and /Core/Src
  - SQIsign did the same thing for the adaptation of verification for PQM4

```
#include <inttypes.h>
#include <mem.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>
#include <time.h>

#include <api.h>
#include <rng.h>
#include <bench_test_arguments.h>
#if defined(TARGET_BIG_ENDIAN)
#include <tutil.h>
#endif
```

```
#include <inttypes.h>
#include "mem.h"
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>
#include <time.h>

#include "api.h"
#include "rng.h"
#include "bench_test_arguments.h"
#if defined(TARGET_BIG_ENDIAN)
#include "tutil.h"
#endif
```

Exemple inside the file example_nistapi.c

Special case

- GMP has 2 version :
  - GMP : normal version
  - GMP mini : smaller version => better for constrain device

Update applications for bare-metal environment

- To solve that => static reading of the code and update it with the HAL
  - Tools.c => use of the clock
  - Randombytes_system.c => use of random

Difficulties

- Not especially hard but :
  - Time consuming
    - Come to a big project
    - Lack of comment
    - Lack of Knowledge about code of SQIsign
    - sqisign_namespace.h
    - Some Bad choice led to redo some parts of the work
  - No real notion of bare metal environment at the beginning
  - Lack of information related to the issues in state of the art

Phase 2 : evaluate SQIsign

NUCLEO-L4R5ZI

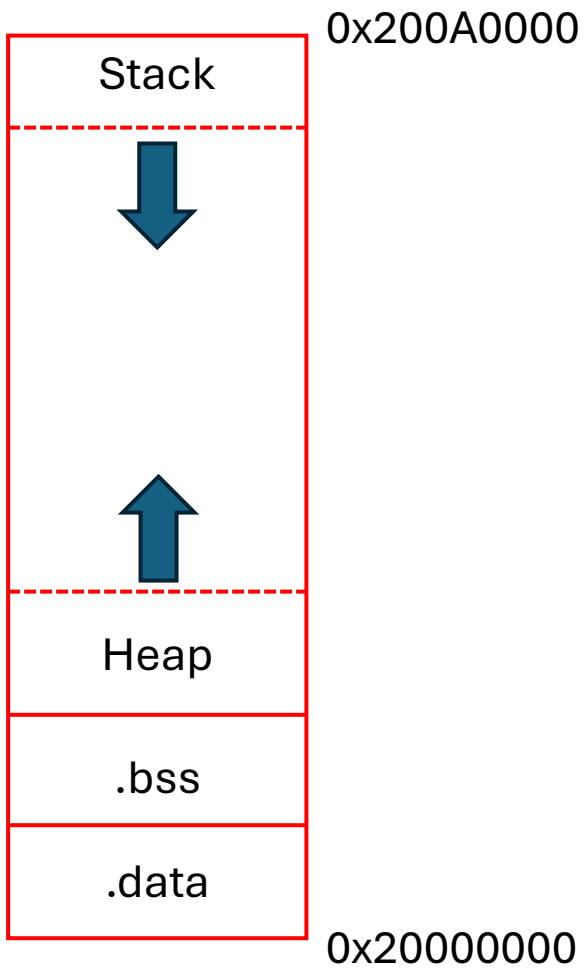
- 2 MB of flash memory
- 640 KB of SRAM
- Selected by SQIsign team

https://www.st.com/en/evaluation-tools/nucleo-l4r5zi.html

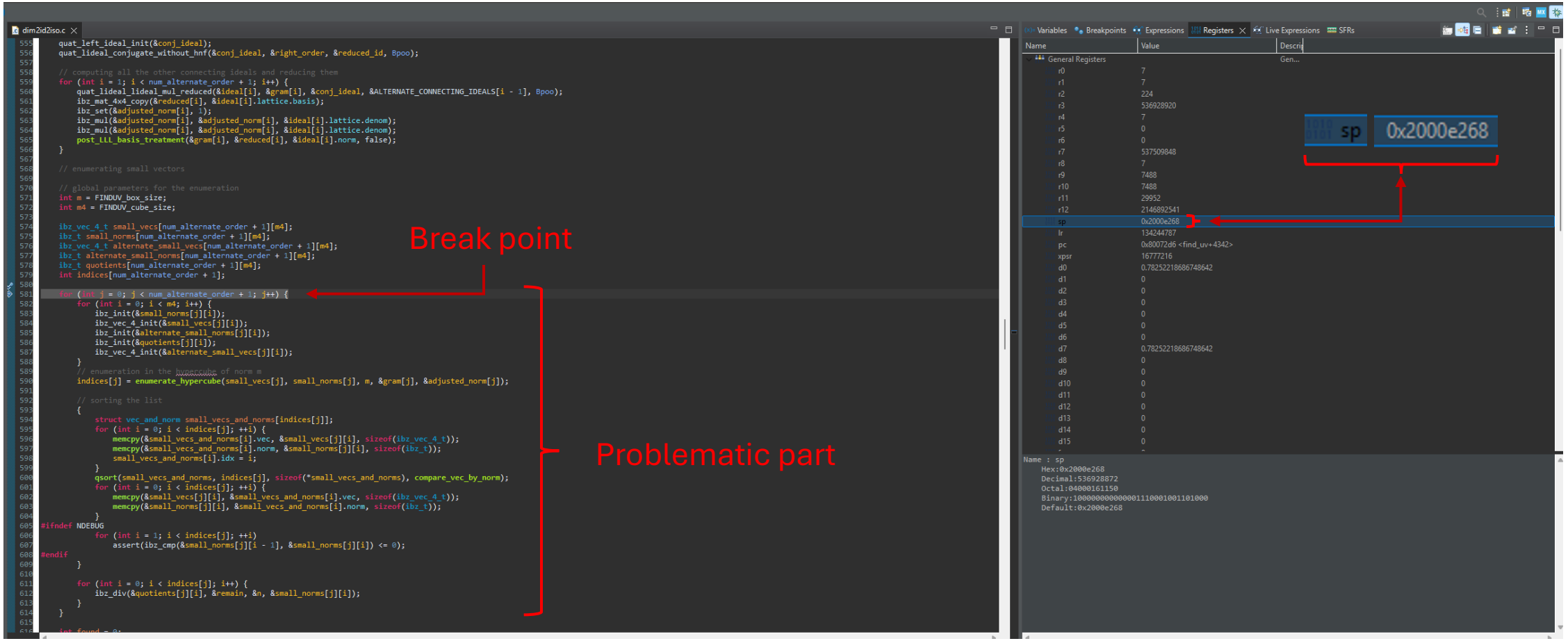
Run the code

- The code crash
  - Ram does not have enough resources
  - Stack grow too much
- Test with a “canary”
  - Technique use in cybersecurity => « something that does not change »
    - Set a global variable initialize with a value, and never touch it during the execution
      - If it change (=> stack overflow because we are in .data section)

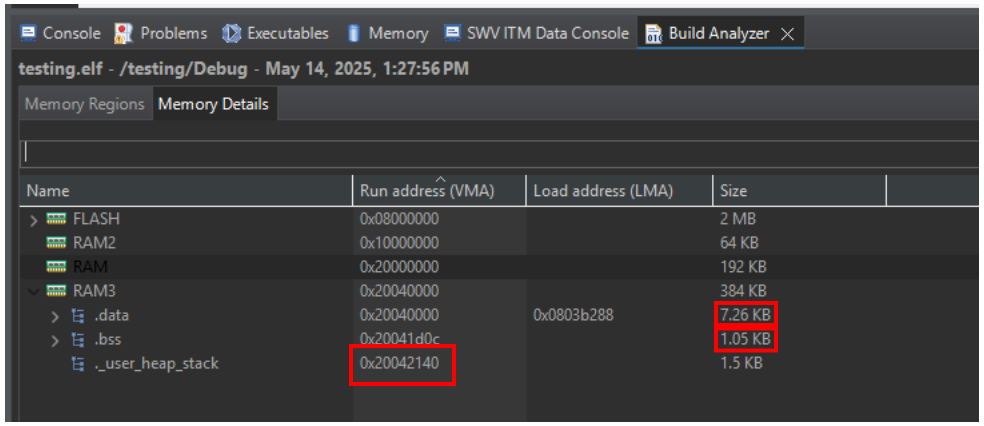
How RAM work

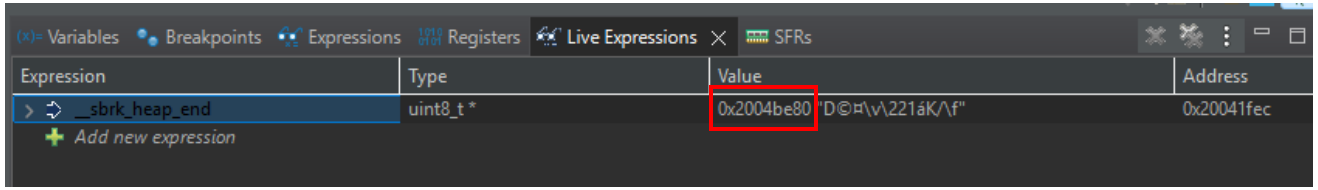


What I did ?


dim2id2iso.c
Break point
Problematic part
Variables
Breakpoints
Expressions
Registers
Live Expressions
SFRs
Name
Value
General Registers
sp
0x2000e268
Name : sp
Hex:0x2000e268
Decimal:536928872
Octal:04000161150
Binary:10000000000000111000100110100 0
Default:0x2000e268

What I did ?


Console
Problems
Executables
Memory
SWV ITM Data Console
Build Analyzer
testing.elf - /testing/Debug - May 14, 2025, 1:27:56 PM
Memory Regions
Memory Details
Name
Run address (VMA)
Load address (LMA)
Size
FLASH
0x08000000
2 MB
RAM2
0x10000000
64 KB
RAM
0x20000000
192 KB
RAM3
0x20040000
384 KB
.data
0x20040000
0x0803b288
7.26 KB
.bss
0x20041d0c
1.05 KB
._user_heap_stack
0x20042140
1.5 KB


Variables
Breakpoints
Expressions
Registers
Live Expressions
SFRs
Expression
Type
Value
Address
__sbrk_heap_end
uint8_t *
0x2004be80
0x20041fec
Add new expression

What I did ?

- Current SP= 0x2000e268
- Start of SP (End of SRAM) = 0x200A0000
- Current Heap pointer = 0x2004be80
- Start of Heap = 0x20042140

- STACK size = Start of SP - Current SP = 583.39 KB
- HEAP “size” = Current Heap pointer - Start of Heap = 39.31 KB
- .bss + .data + HEAP + STACK = ~ 631 KB (out of 640 KB provide by board)

Algorithm 4.1 SQIsign.KeyGen( )

Output: Secret key $\mathtt{sk}$ and public key $\mathtt{pk}$.

while $\mathtt{true}$ **do**
  $I_{\mathsf{sk}} \leftarrow \mathsf{RandomIdealGivenNorm}(D_{\mathsf{mix}}, \mathtt{true})$
  try
    $I_{\mathsf{sk}} \leftarrow \mathsf{RandomEquivalentPrimeIdeal}(I_{\mathsf{sk}})$
    $E_{\mathsf{pk}}, \varphi_{\mathsf{sk}}(P_0), \varphi_{\mathsf{sk}}(Q_0) \leftarrow \mathsf{IdealToIsogeny}(I_{\mathsf{sk}})$
  except
    continue
  $P_{\mathsf{pk}}, Q_{\mathsf{pk}}, \mathsf{hint}_{\mathsf{pk}} \leftarrow \mathsf{TorsionBasisToHint}(E_{\mathsf{pk}})$
  $\mathbf{M}_{\mathsf{sk}} \leftarrow \mathsf{ChangeOfBasis}_{2^f}(E_{\mathsf{pk}}, (\varphi_{\mathsf{sk}}(P_0), \varphi_{\mathsf{sk}}(Q_0)), (P_{\mathsf{pk}}, Q_{\mathsf{pk}}))$
  $\mathsf{pk} \leftarrow (E_{\mathsf{pk}}, \mathsf{hint}_{\mathsf{pk}})$
  $\mathsf{sk} \leftarrow (E_{\mathsf{pk}}, \mathsf{hint}_{\mathsf{pk}}, I_{\mathsf{sk}}, \mathbf{M}_{\mathsf{sk}})$
  return sk, pk

Algorithm 3.13 IdealToIsogeny(I)

Input: A left $\mathcal{O}_0$-ideal I.
Output: The codomain E_I of φ_I, $\varphi_I(P_0)$, and $\varphi_I(Q_0)$.

$u, v, e, s, t, \beta_1, \beta_2 \leftarrow \mathsf{SuitableIdeals}(I)$
$d_1 \leftarrow \mathrm{nrd}(\beta_1)/\mathrm{nrd}(\overline{J_s}I)$
$d_2 \leftarrow \mathrm{nrd}(\beta_2)/\mathrm{nrd}(\overline{J_t}I)$ // $ud_1 + vd_2 = 2^e$, with $\gcd(ud_1, vd_2) = 1$
$E_u, \varphi_u(P_s), \varphi_u(Q_s) \leftarrow \mathsf{FixedDegreeIsogeny}(s, u)$ // $\varphi_u : E_s \to E_u$ is an isogeny of degree u
$E_v, \varphi_v(P_t), \varphi_v(Q_t) \leftarrow \mathsf{FixedDegreeIsogeny}(t, v)$ // $\varphi_v : E_t \to E_v$ is an isogeny of degree v
$[P, Q]^T \leftarrow \left[\frac{1}{\mathrm{nrd}(I)\,\mathrm{nrd}(J_t)}\right] \mathbf{M}_{\overline{\beta_1}} \mathbf{M}_{\beta_2} [\varphi_v(P_t),\ \varphi_v(Q_t)]^T$
$K_P \leftarrow [2^{f-e}]([d_1]\varphi_u(P_s), P)$
$K_Q \leftarrow [2^{f-e}]([d_1]\varphi_u(Q_s), Q)$
$E \times E', [(P, P'), (Q, Q')] \leftarrow \mathsf{Isogeny22Chain}(K_P, K_Q, [(\varphi_u(P_s), 0_{E_v}), (\varphi_u(Q_s), 0_{E_v})])$
if $e_{2^f}(P, Q) = e_{2^f}(P_s, Q_s)^{u^2 d_1}$ **then** // We identify the right curve E_I by ensuring it is d_1-isogeneous to E_0
  $E_I \leftarrow E$, $P_I \leftarrow P$, and $Q_I \leftarrow Q$
else
  $E_I \leftarrow E'$, $P_I \leftarrow P'$, and $Q_I \leftarrow Q'$
$[P_I, Q_I]^T \leftarrow \left[\frac{1}{ud_1}\right] \mathbf{M}_{\beta_1} [P_I,\ Q_I]^T$
return $E_I, (P_I, Q_I)$

Algorithm 3.16 SuitableIdeals(I)

Input: A left $\mathcal{O}_0$-ideal I.
Output: Positive integers u, v, e, indices s, t, and $\beta_1 \in \overline{J_s}I, \beta_2 \in \overline{J_t}I$ such that $ud_1 + vd_2 = 2^e$, $\gcd(ud_1, vd_2) = 1$, and $e \le f$, where $d_1 = \mathrm{nrd}(\beta_1)/\mathrm{nrd}(\overline{J_s}I)$, $d_2 = \mathrm{nrd}(\beta_2)/\mathrm{nrd}(\overline{J_t}I)$.

for t **from** 0 **up to** n_{orders} **do**
  Let $(b_1, \ldots, b_4)$ be a basis of $\overline{J_t}I$
  Compute the Gram matrix $\mathbf{G} = (\langle b_r, b_s \rangle)_{1 \le r,s \le 4}$
  $(\alpha_1, \ldots, \alpha_4), \_ \leftarrow \mathsf{L2}_{\delta,\eta}((b_1, \ldots, b_4), \mathbf{G})$
  $L_t \leftarrow \left[\sum_{s=1}^4 x_s \alpha_s \;\middle|\; (x_1, \ldots, x_4) \in [-m, \ldots, m]\right]$ for $m = \mathtt{FINDUV\_box\_size}$
  Sort L_t by increasing norm.
for $(s, t) \in [0, \ldots, n_{\text{orders}}]^2$ **do** // by increasing lexicographic order
  for $(\beta_1, \beta_2) \in L_s \times L_t$ **do** // by increasing lexicographic order
    $d_1 \leftarrow \mathrm{nrd}(\beta_1)/\mathrm{nrd}(\overline{J_s}I)$ and $d_2 \leftarrow \mathrm{nrd}(\beta_2)/\mathrm{nrd}(\overline{J_t}I)$
    if $d_1 \equiv 1 \pmod 2$ and $d_2 \equiv 1 \pmod 2$ and $\gcd(d_1, d_2) = 1$ **then**
      $u \leftarrow 2^f d_1^{-1} \pmod{d_2}$
      $v \leftarrow (2^f - ud_1)/d_2$
      if $v > 0$ **then**
        $e \leftarrow \mathsf{DyadicValuation}(u)$
        $u \leftarrow u/2^e$, $v \leftarrow v/2^e$, and $e \leftarrow f - e$
        return $u, v, e, s, t, \beta_1, \beta_2$
raise Exception("SuitableIdeals failed")

Algorithm 4.1 SQIsign.KeyGen( )

Output: Secret key sk and public key pk.

while `true` **do**
  $I_{\mathsf{sk}} \leftarrow \mathsf{RandomIdealGivenNorm}(D_{\mathsf{mix}}, \texttt{true})$
  try
    $I_{\mathsf{sk}} \leftarrow \mathsf{RandomEquivalentPrimeIdeal}(I_{\mathsf{sk}})$
    $E_{\mathsf{pk}}, \varphi_{\mathsf{sk}}(P_0), \varphi_{\mathsf{sk}}(Q_0) \leftarrow \mathsf{IdealToIsogeny}(I_{\mathsf{sk}})$
  except
    continue
  $P_{\mathsf{pk}}, Q_{\mathsf{pk}}, \mathsf{hint}_{\mathsf{pk}} \leftarrow \mathsf{TorsionBasisToHint}(E_{\mathsf{pk}})$
  $\mathbf{M}_{\mathsf{sk}} \leftarrow \mathsf{ChangeOfBasis}_{2^f}(E_{\mathsf{pk}}, (\varphi_{\mathsf{sk}}(P_0), \varphi_{\mathsf{sk}}(Q_0)), (P_{\mathsf{pk}}, Q_{\mathsf{pk}}))$
  $\mathsf{pk} \leftarrow (E_{\mathsf{pk}}, \mathsf{hint}_{\mathsf{pk}})$
  $\mathsf{sk} \leftarrow (E_{\mathsf{pk}}, \mathsf{hint}_{\mathsf{pk}}, I_{\mathsf{sk}}, \mathbf{M}_{\mathsf{sk}})$
  return sk, pk

Algorithm 3.13 IdealToIsogeny(I)

Input: A left $\mathcal{O}_0$-ideal I.
Output: The codomain E_I of φ_I, $\varphi_I(P_0)$, and $\varphi_I(Q_0)$.

$u, v, e, s, t, \beta_1, \beta_2 \leftarrow \mathsf{SuitableIdeals}(I)$
$d_1 \leftarrow \mathrm{nrd}(\beta_1)/\mathrm{nrd}(\overline{J_s}I)$
$d_2 \leftarrow \mathrm{nrd}(\beta_2)/\mathrm{nrd}(\overline{J_t}I)$ // $ud_1 + vd_2 = 2^e$, with $\gcd(ud_1, vd_2) = 1$
$E_u, \varphi_u(P_s), \varphi_u(Q_s) \leftarrow \mathsf{FixedDegreeIsogeny}(s, u)$ // $\varphi_u : E_s \to E_u$ is an isogeny of degree u
$E_v, \varphi_v(P_t), \varphi_v(Q_t) \leftarrow \mathsf{FixedDegreeIsogeny}(t, v)$ // $\varphi_v : E_t \to E_v$ is an isogeny of degree v
$[P, Q]^T \leftarrow \left[\frac{1}{\mathrm{nrd}(I)\,\mathrm{nrd}(J_t)}\right] \mathbf{M}_{\overline{\beta_1}} \mathbf{M}_{\beta_2} [\varphi_v(P_t),\ \varphi_v(Q_t)]^T$
$K_P \leftarrow [2^{f-e}]([d_1]\varphi_u(P_s), P)$
$K_Q \leftarrow [2^{f-e}]([d_1]\varphi_u(Q_s), Q)$
$E \times E', [(P, P'), (Q, Q')] \leftarrow \mathsf{Isogeny22Chain}(K_P, K_Q, [(\varphi_u(P_s), 0_{E_v}), (\varphi_u(Q_s), 0_{E_v})])$
if $e_{2^f}(P, Q) = e_{2^f}(P_s, Q_s)^{u^2 d_1}$ **then** // We identify the right curve E_I by ensuring it is d_1-isogeneous to E_0
  $E_I \leftarrow E$, $P_I \leftarrow P$, and $Q_I \leftarrow Q$
else
  $E_I \leftarrow E'$, $P_I \leftarrow P'$, and $Q_I \leftarrow Q'$
$[P_I, Q_I]^T \leftarrow \left[\frac{1}{ud_1}\right] \mathbf{M}_{\beta_1} [P_I,\ Q_I]^T$
return $E_I, (P_I, Q_I)$

Algorithm 3.16 SuitableIdeals(I)

Input: A left $\mathcal{O}_0$-ideal I.
Output: Positive integers u, v, e, indices s, t, and $\beta_1 \in \overline{J_s}I, \beta_2 \in \overline{J_t}I$ such that $ud_1 + vd_2 = 2^e$, $\gcd(ud_1, vd_2) = 1$, and $e \le f$, where $d_1 = \mathrm{nrd}(\beta_1)/\mathrm{nrd}(\overline{J_s}I)$, $d_2 = \mathrm{nrd}(\beta_2)/\mathrm{nrd}(\overline{J_t}I)$.

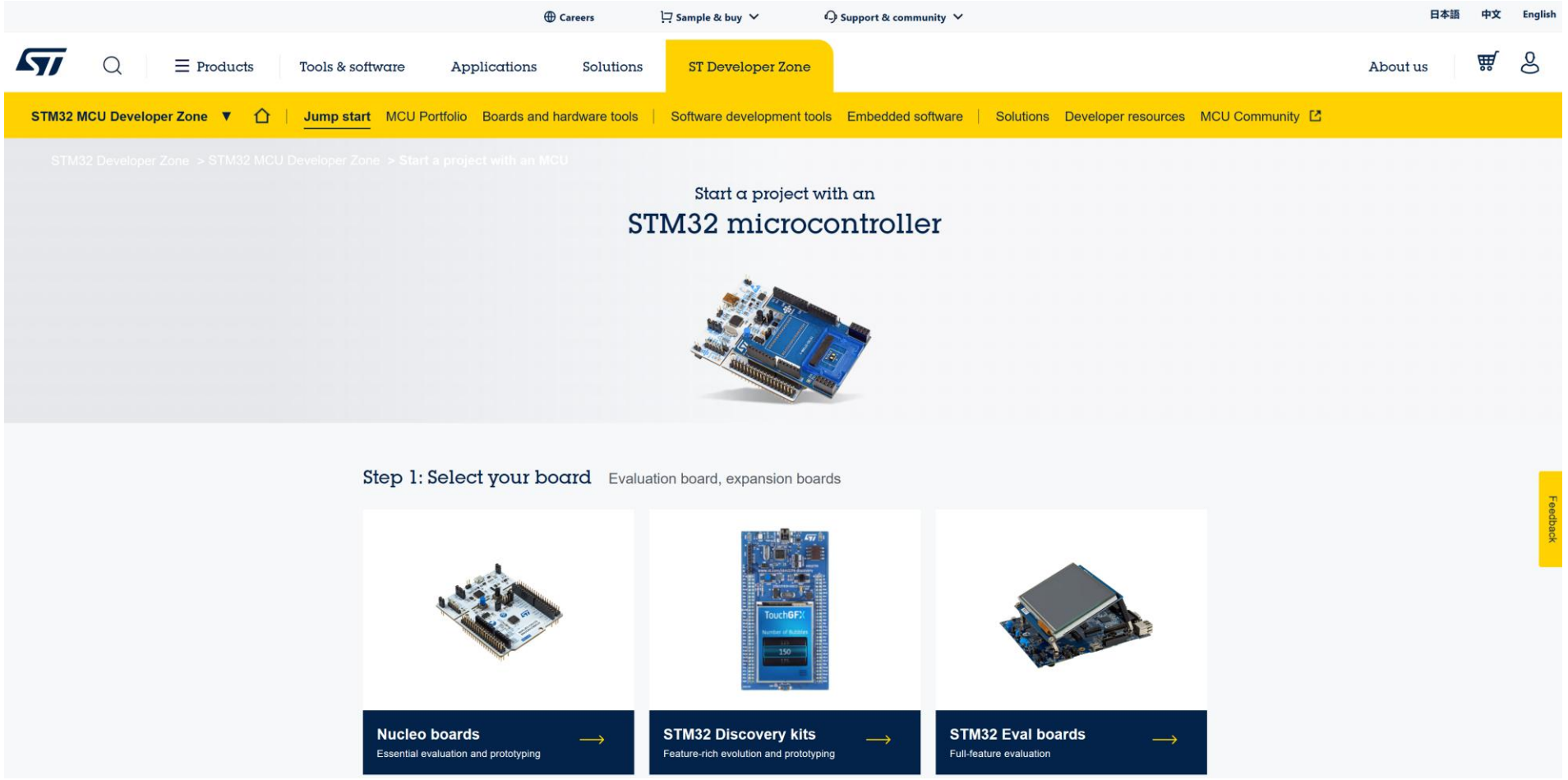
for t **from** 0 **up to** n_{orders} **do**
  Let $(b_1, \ldots, b_4)$ be a basis of $\overline{J_t}I$
  Compute the Gram matrix $\mathbf{G} = (\langle b_r, b_s \rangle)_{1 \le r,s \le 4}$
  $(\alpha_1, \ldots, \alpha_4), \_ \leftarrow \mathsf{L2}_{\delta,\eta}((b_1, \ldots, b_4), \mathbf{G})$
  $L_t \leftarrow \left[\sum_{s=1}^4 x_s \alpha_s \;\middle|\; (x_1, \ldots, x_4) \in [-m, \ldots, m]\right]$ for $m =$ `FINDUV_box_size`
  Sort L_t by increasing norm.
for $(s, t) \in [0, \ldots, n_{\text{orders}}]^2$ **do** // by increasing lexicographic order
  for $(\beta_1, \beta_2) \in L_s \times L_t$ **do** // by increasing lexicographic order
    $d_1 \leftarrow \mathrm{nrd}(\beta_1)/\mathrm{nrd}(\overline{J_s}I)$ and $d_2 \leftarrow \mathrm{nrd}(\beta_2)/\mathrm{nrd}(\overline{J_t}I)$
    if $d_1 \equiv 1 \pmod 2$ and $d_2 \equiv 1 \pmod 2$ and $\gcd(d_1, d_2) = 1$ **then**
      $u \leftarrow 2^f d_1^{-1} \pmod{d_2}$
      $v \leftarrow (2^f - ud_1)/d_2$
      if $v > 0$ **then**
        $e \leftarrow \mathsf{DyadicValuation}(u)$
        $u \leftarrow u/2^e$, $v \leftarrow v/2^e$, and $e \leftarrow f - e$
        return $u, v, e, s, t, \beta_1, \beta_2$
raise Exception("SuitableIdeals failed")

Difficulties

- Have enough skills thanks to my studies in cybersecurity with heap, stack, … but :
  - New tool (=> debugger STM32CubeIde)
  - Code run really slowly, approximative 5 minutes to reach the point (=> run multiple time to understand the situation)
  - Make work Printf

There is another way to evaluate SQIsign and potentially see if all my adaptation works ?

Phase 3



https://www.st.com/content/st_com/en/stm32-mcu-developer-zone/start-a-project-with-an-mcu.html

Phase 3

NUCLEO-L4R5ZI

https://www.st.com/en/evaluation-tools/nucleo-l4r5zi.html

STM32F429I-DISC1

https://www.st.com/en/evaluation-tools/32f429idiscovery.html

Phase 3

NUCLEO-L4R5ZI

- 2 MB of flash memory
- 640 KB of SRAM

STM32F429I-DISC1

- 2 MB of flash memory
- 256 KB of SRAM
- 8 MB of SDRAM

  (=> frame buffer for LCD screen)

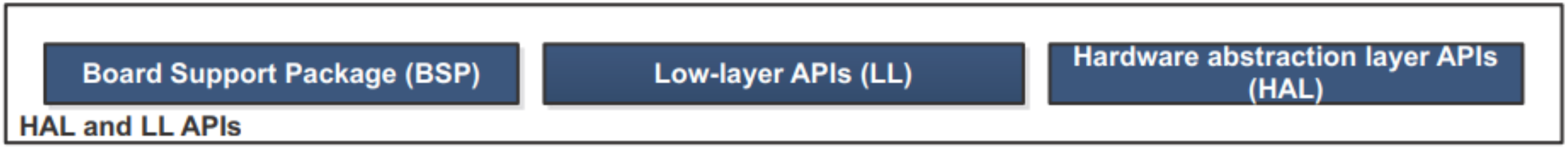
Why not use the SDRAM as base for my stack ?

Phase 3

- Step 1 : initialize the SDRAM
- Step 2 : move the stack

Initialization of the SDRAM

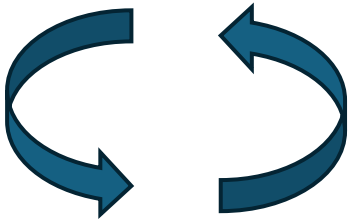
Not so simple, not like « plug and play » when we have OS



https://www.st.com/resource/en/user_manual/um2298-stm32cube-bsp-drivers-development-guidelines-stmicroelectronics.pdf

Move the stack

```
/* Highest address of the user mode stack */
_estack = ORIGIN(RAM) + LENGTH(RAM);

/* Memories definition */
MEMORY
{
  CCMRAM (xrw) : ORIGIN = 0x10000000, LENGTH = 64K
  RAM    (xrw) : ORIGIN = 0x20000000, LENGTH = 192K
  FLASH  (rx)  : ORIGIN = 0x8000000,  LENGTH = 2048K
  SDRAM  (xrw) : ORIGIN = 0xD0000000, LENGTH = 8192K
}
```



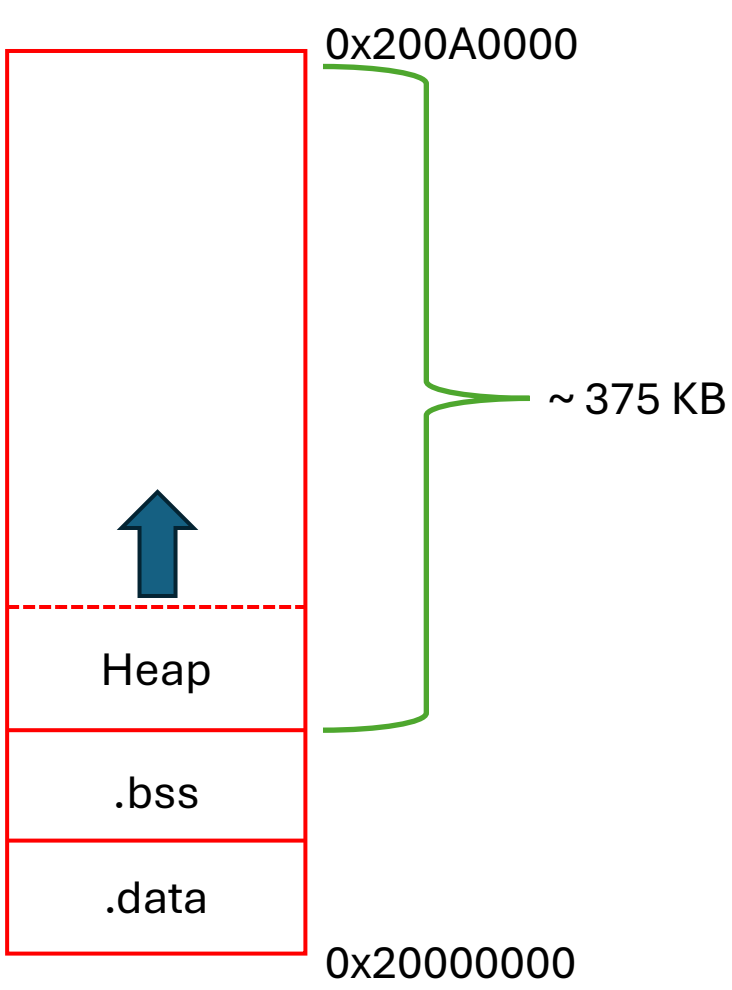
```
int main(void)
{

  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */


  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  //MX_FMC_Init();
  BSP_SDRAM_Init();
  MX_TIM1_Init();
  MX_USART1_UART_Init();
  MX_RNG_Init();
  MX_I2C3_Init();
  MX_SPI5_Init();
```

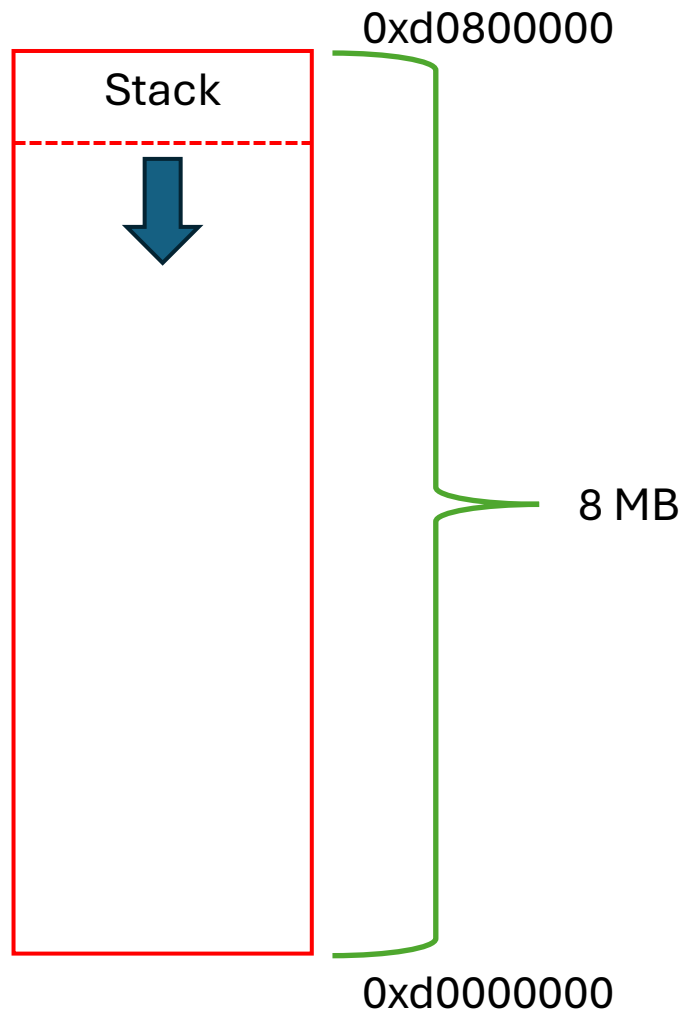
Move the stack

```
BSP_SDRAM_Init();
MX_TIM1_Init();
MX_USART1_UART_Init();
MX_RNG_Init();
MX_I2C3_Init();
MX_SPI5_Init();
/* USER CODE BEGIN 2 */
SDRAM_Clear();
__asm volatile (
      "msr msp, %0\n"
      :
      : "r"(&__stack_end__)
      :
  );
/* USER CODE END 2 */
```

Move manually SP to the end of the SDRAM

State of RAM


0x200A0000
~ 375 KB
Heap
.bss
.data
0x20000000


0xd0800000
Stack
8 MB
0xd0000000

First victory

=> Key generation works

Difficulties

- Begin to understand the meaning of bare metal environment in term of hardware
- Lack of documentation over internet to move the stack

Phase 4 : The bug

- Code randomly crash at any routine :
  - Key generation
  - Signature
  - Verification
- Never see a bug like that before
- 3 Hypothesis :
  - Stack overflow
  - Heap overflow
  - Buffer overflow with file FP or use of GMP

Phase 4 : The bug

- 3 Hypothesis :
  - Stack overflow (?)
  - Heap overflow (?)
  - Buffer overflow with file FP or use of GMP (?)

Reading code + ChatGPT

Real problem : The BSP

- Adapted be normally used as frame buffer for LCD
  - Must remove LTDC + DMA2D (hardware for LCD)
  - Clean BSP
  - Increase the clock system => the BSD adapted for a clock system of 180 MHz

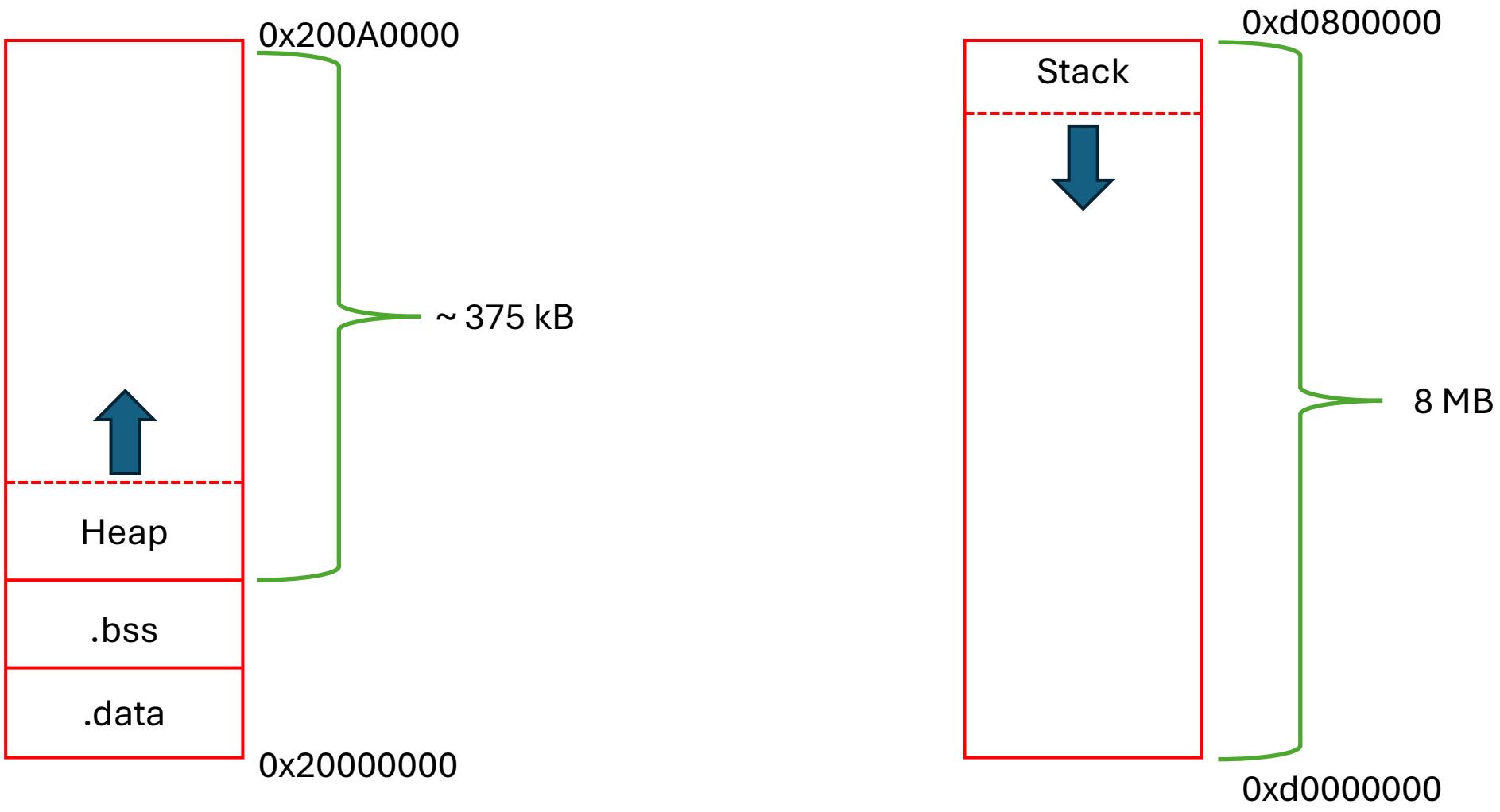
Difficulties

- Take a very long time to solve (time to run the code a least once ~ 33 minutes)
- Lot of new notion that I never see during my studies (hardware/electronics)
- New tools (STM32 programmer and monitor)

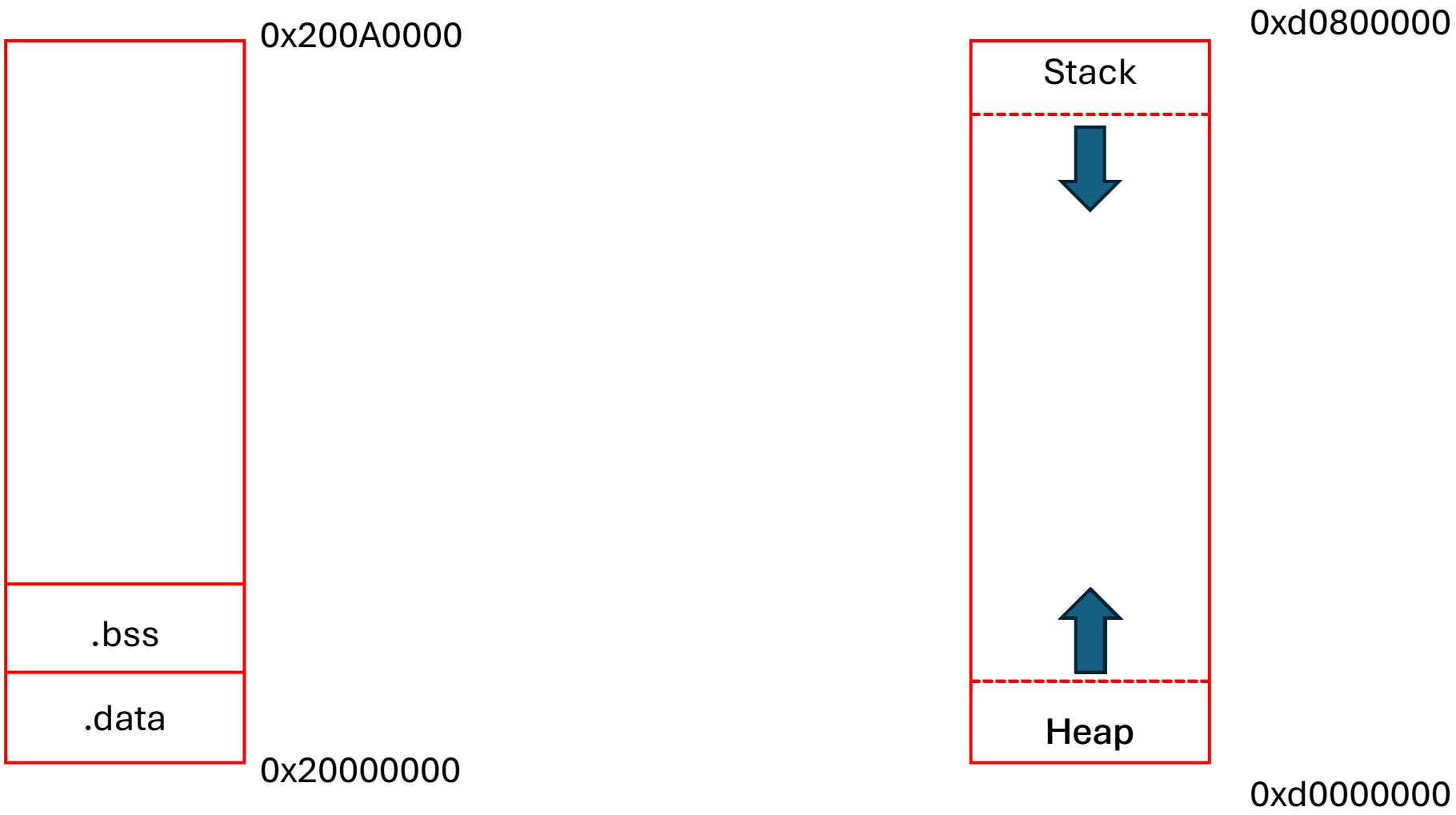
Phase 5 : Evaluate SQIsign with all level of difficulties

- Level 1 
- Level 3 
- Level 5 

State of RAM


0x200A0000
~ 375 kB
Heap
.bss
.data
0x20000000
0xd0800000
Stack
8 MB
0xd0000000

State of RAM


0x200A0000
.bss
.data
0x20000000
0xd0800000
Stack
Heap
0xd0000000

Phase 5 : Evaluate SQIsign with all level of difficulties

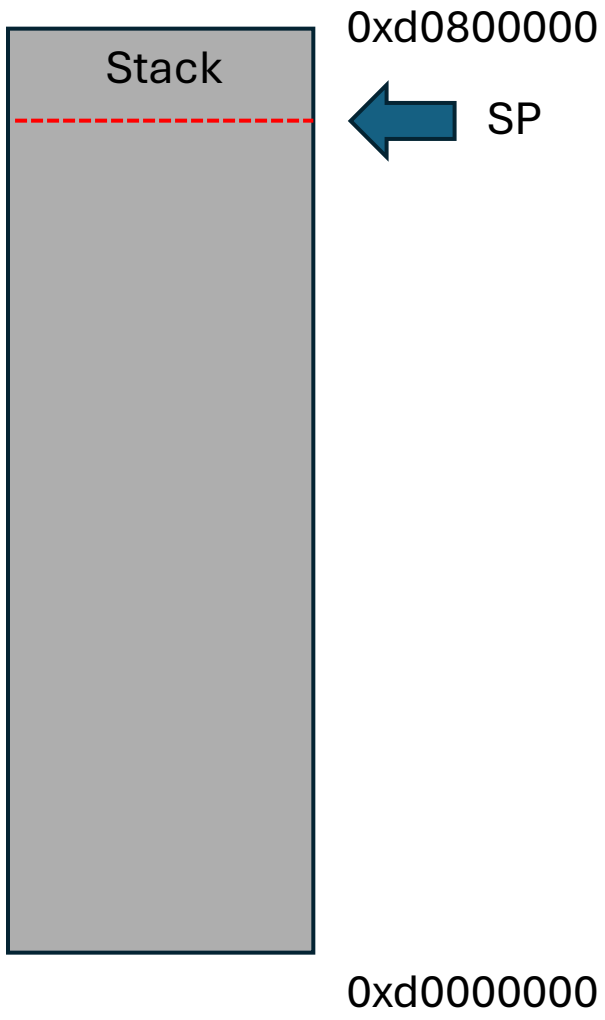
- Level 1 ✓
- Level 3 ✓
- Level 5 ✓

Phase 6 : Data gathering

- First time after cycle
- Heap only grow, never shrink
- Stack => technique of « stack painting »

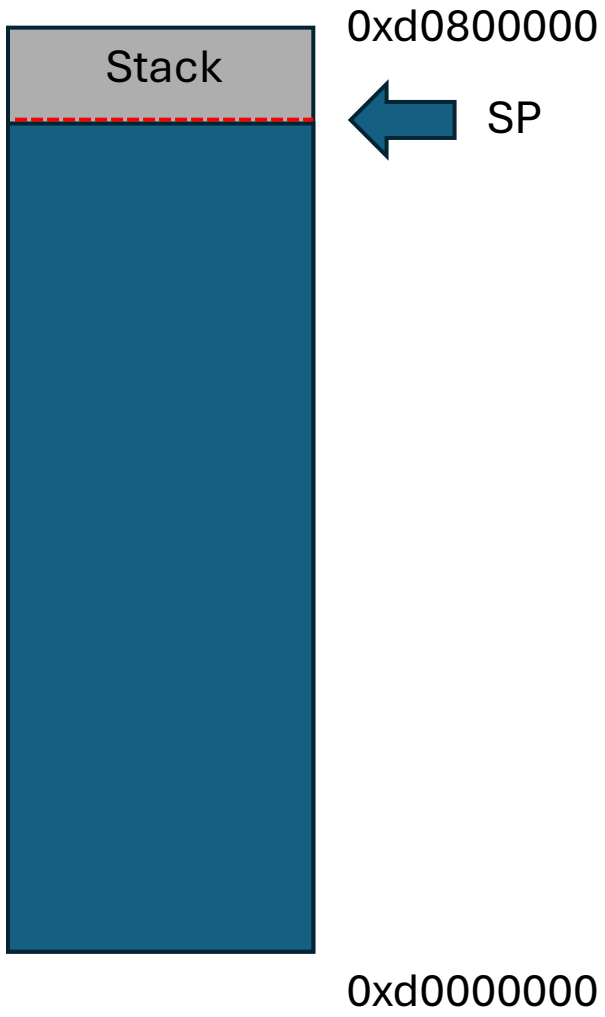
Stack painting

1) Fill the stack from lower address to Stack Point with a specific value (example : 0xAB)



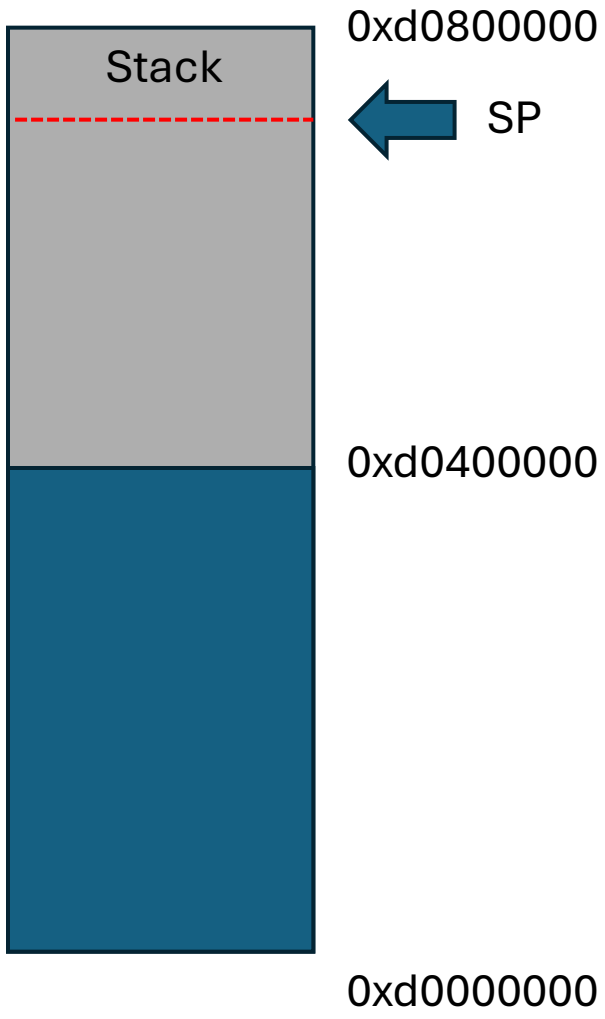
Stack painting (simplified)

1) Fill the stack from lower address to Stack Point with a specific value (example : 0xAB)
2) Let the code run



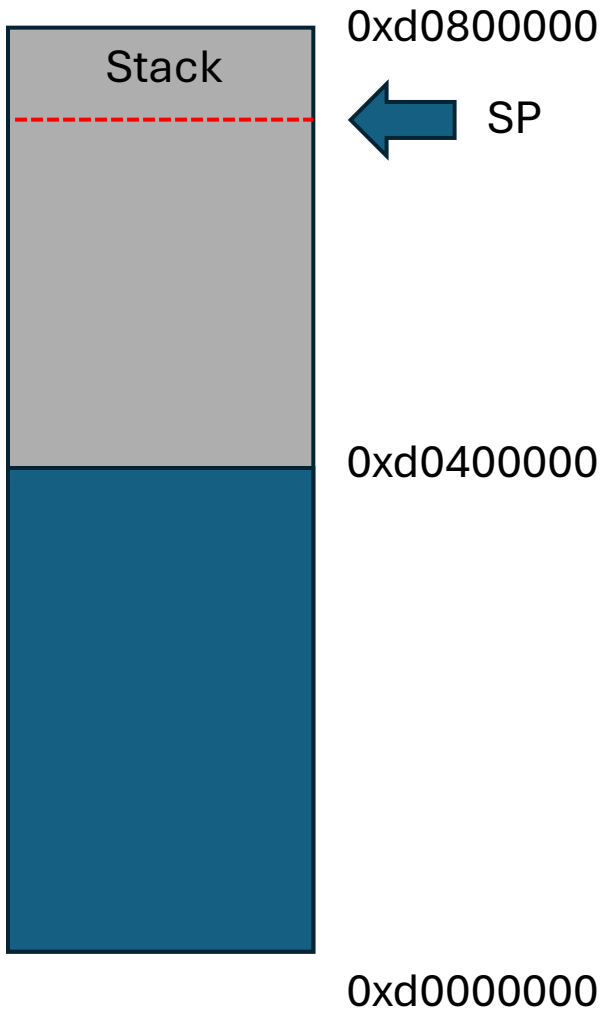
Stack painting (simplified)

1) Fill the stack from lower address to Stack Point with a specific value (example : 0xAB)
2) Let the code run



Stack painting (simplified)

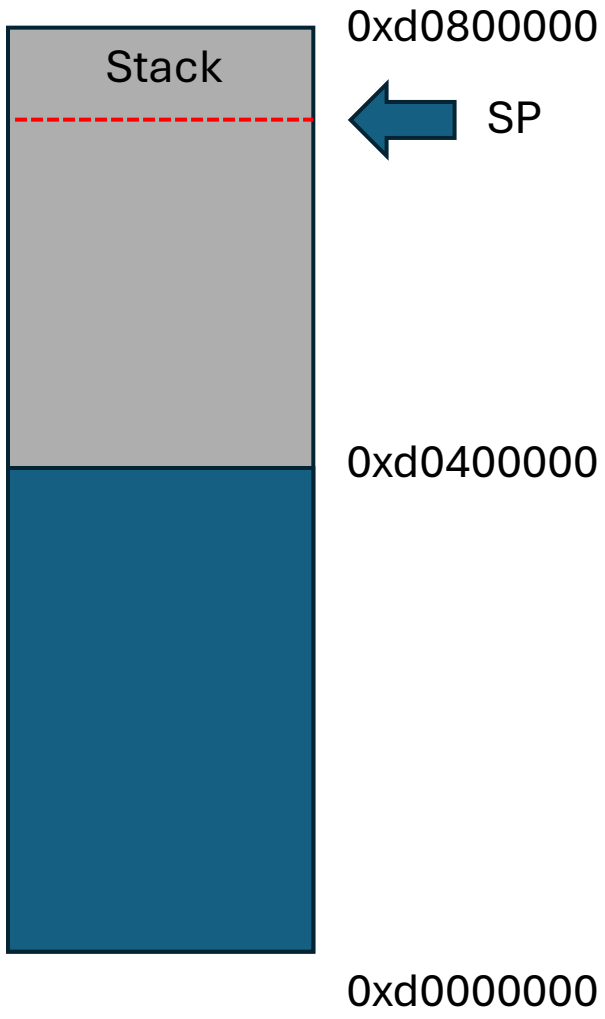
1) Fill the stack from lower address to Stack Point with a specific value (example : 0xAB)
2) Let the code run
3) From lower address begin to observe a which address we do not find anymore our specific value



Stack painting (simplified)

1) Fill the stack from lower address to Stack Point with a specific value (example : 0xAB)
2) Let the code run
3) From lower address begin to observe a which address we do not find anymore our specific value
4) “Begin of the stack” - “Address found” = stack usage

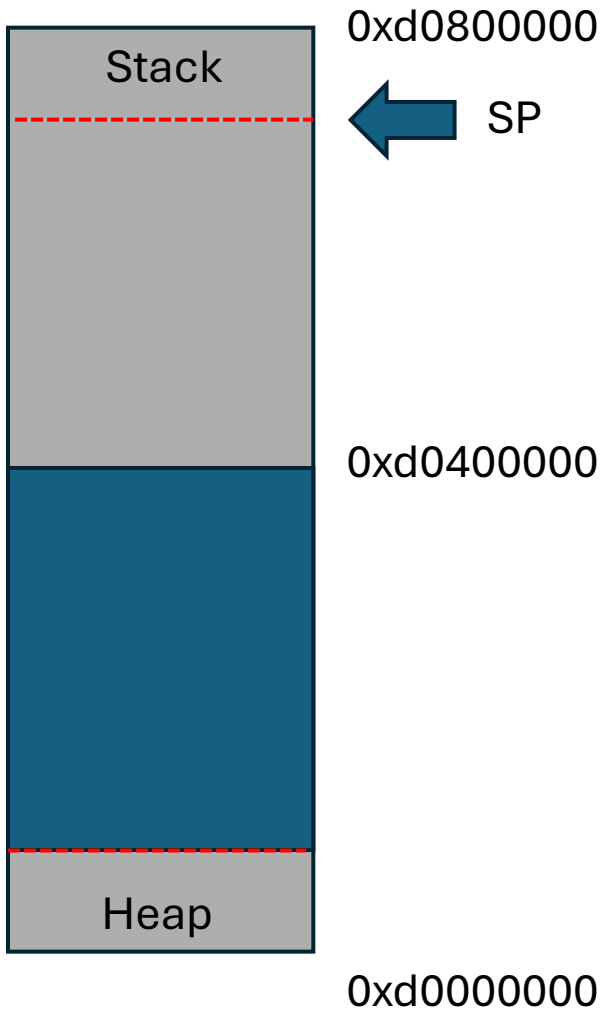
0xd0800000 - 0xd0400000 = 4 MB



Stack painting (With heap)

1) Fill the stack from **address of higher point of heap** to Stack Point with a specific value (example : 0xAB)
2) Let the code run
3) From **address of higher point of heap** begin to observe a which address we do not find anymore our specific value
4) “Begin of the stack” - “Address found” = stack usage

0xd0800000 - 0xd0400000 = 4 MB



Data Analysis

- Size code :
  - Level 1 : 251.02 KB | .bss : 1.62 KB | .data : 7.27 KB
  - Level 3 : 302.76 KB | .bss : 1.71 KB | .data : 12.3 KB
  - Level 5 : 363.30 KB | .bss : 1.61 KB | .data : 14.23 KB

Level 1

	Key Generation			Sign			Verification				
	Time	Cycle	Stack size	Time	Cycle	Stack size	Time	Cycle	Stack size		Heap usage
	m	MC	KB	m	MC	KB	m	MC	B		KB
Average	11.8767048	2511.77192	591.055158	21.0894416	2022.48404	595.661096	0.852116	612.951765	37324		91.48578
STD	5.08582841	1380.35171	0.88579772	3.63281444	1319.18558	0.84102802	0.01648659	178.072344	0		1.35058852
MAX	31.04008	4239.47333	592.33984	27.6488	4121.19041	596.56641	0.87365	845.48542	37324		92.47656
MIN	7.45138	15.74904	590.41797	15.07465	2.04906	594.66797	0.82792	351.63237	37324		86.48047

Level 3

	Key Generation			Sign			Verification				
	Time	Cycle	Stack size	Time	Cycle	Stack size	Time	Cycle	Stack size		Heap usage
	m	MC	MB	m	MC	MB	m	MC	KB		KB
Average	36.07258	2768.35932	2.47372188	69.6412738	2117.096	2.48011188	2.85425375	761.150262	60.45703		421.982909
STD	11.5108994	924.153136	0.00103319	13.7229398	1443.0889	0.00076418	0.05187097	560.234755	2.2015E-14		20.6294427
MAX	64.19558	4115.71065	2.47533	96.6169	4080.44468	2.48066	2.92553	1531.00593	60.45703		437.46875
MIN	24.5804	1375.67845	2.47295	48.64852	23.12968	2.47862	2.7955	126.56757	60.45703		384.20703

Level 5

	Key Generation			Sign			Verification				
	Time	Cycle	Stack size	Time	Cycle	Stack size	Time	Cycle	Stack size		Heap usage
	m	MC	MB	m	MC	MB	m	MC	KB		KB
Average	78.8275517	2365.68384	2.176985	154.731542	2505.88312	2.18348167	6.07347667	1884.90245	79.71484		402.822915
STD	13.8266964	1293.73328	0.00113885	27.7205353	1272.88002	0.00110465	0.12399968	1534.92315	0		7.30196233
MAX	97.77285	3679.82585	2.17791	191.66892	4145.14787	2.1845	6.20428	4001.89791	79.71484		405.86328
MIN	56.96618	716.90664	2.17553	112.77808	676.30501	2.18246	5.9381	67.51296	79.71484		387.91797

Data criticism

- Technologie : SDRAM slower than SRAM (x10 to x100)
  - Increase time
  - Increase number cycle
  - (does not change the stack and heap)
  - Very bad optimization
- Compare to the work of team SQIsign :
  - data small change in term of stack usage
  - Not the same code for verification

Performance data for verification only using a pure C implementation, running on an ARM *Cortex-M4* CPU:

parameter set	clock cycles	stack usage	code size
NIST - I	123 megacycles	31 kilobytes	40 kilobytes
NIST - III	375 megacycles	50 kilobytes	44 kilobytes
NIST - V	751 megacycles	64 kilobytes	46 kilobytes

Future work

- Compare/evaluate with and without FPU
- Adapt the work to PQM4
- Improve SQIsign to use less RAM resource
- Improve hardware configuration
- Evaluate if the code work between M4 and 64 bits architecture
- Update verification with the recent work of SQIsign team

END

Printf

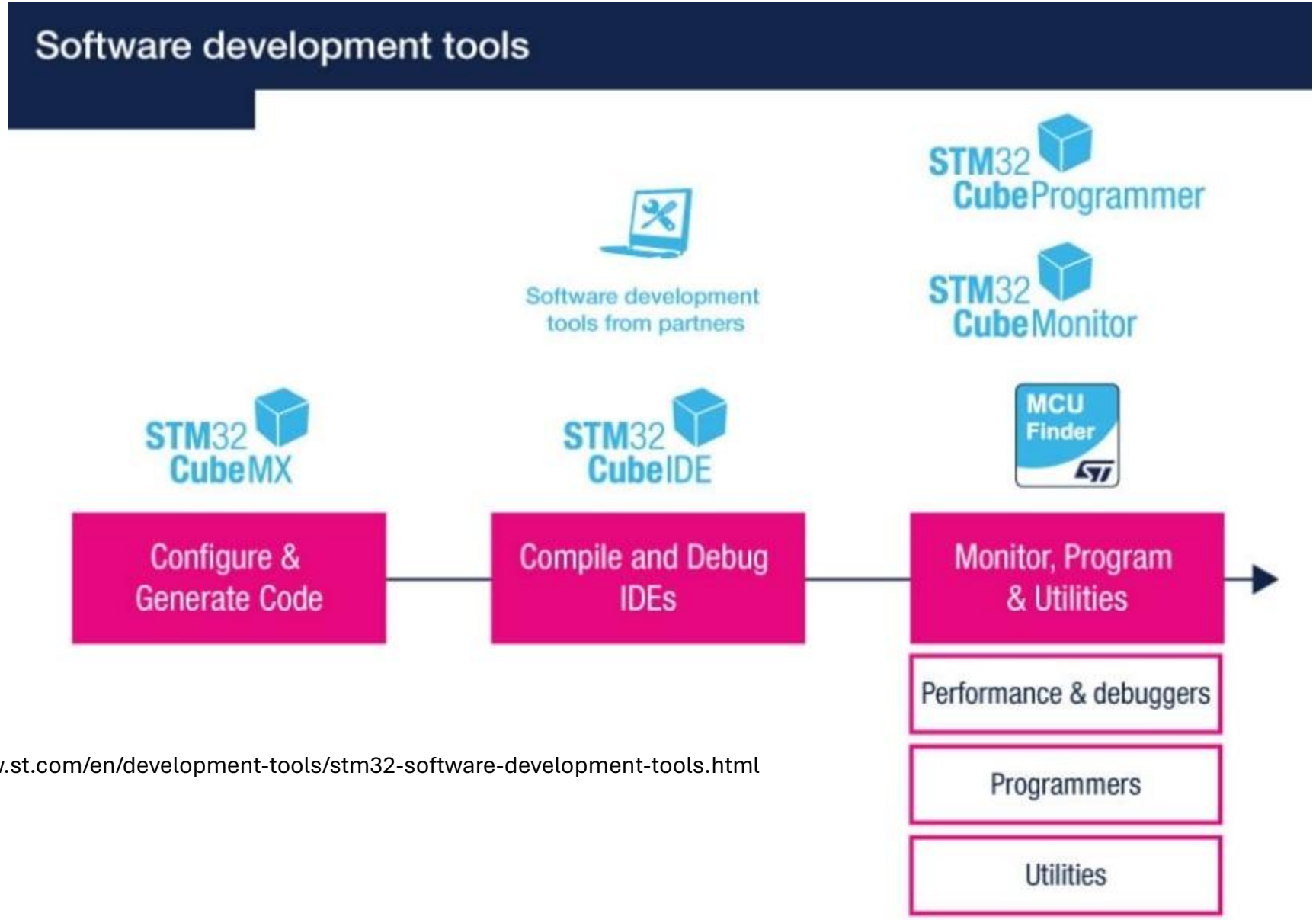
- 2 Solutions
  - use Serial debugger (output on cube ide)
  - Use UART (output on external program Tera Term 5)
- Serial debugger => initial solution
- UART => final solution to gather datas


STM32
CubeIDE


STM32
CubeProgrammer

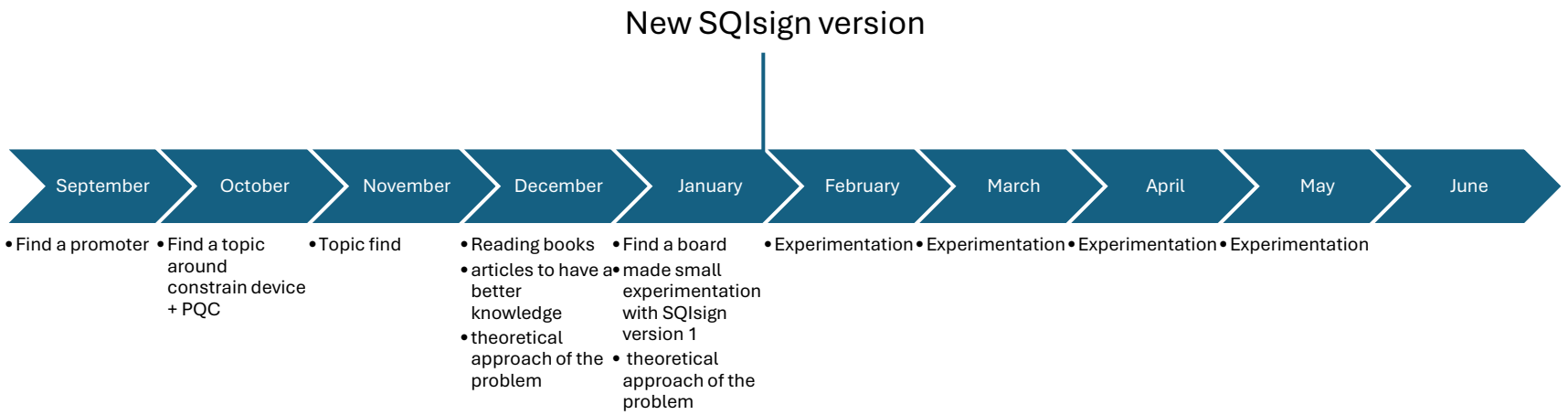

STM32
CubeMonitor


STM32
CubeMX



https://www.st.com/en/development-tools/stm32-software-development-tools.html

Timeline (personal work)



Timeline (NIST)

- Final date for proposition
- End round 1
- End round 2

Go to 32bits to 64bits

- Modifie the Fp file
- Modifie inside macroM4.h, radix 32 to radix 64

Evaluate the good work between M4 and 64 bits architecture

ARM CORTEX M4 datasheet

Floating-Point Unit	Optional single precision floating-point unit IEEE 754 compliant

https://www.st.com/en/evaluation-tools/nucleo-l4r5zi.html

GMP

Floating Point Mode

On some systems, the hardware floating point has a control mode which can set all operations to be done in a particular precision, for instance single, double or extended on x86 systems (x87 floating point). The GMP functions involving a `double` cannot be expected to operate to their full precision when the hardware is in single precision mode. Of course this affects all code, including application code, not just GMP.

https://gmplib.org/gmp-man-6.3.0.pdf

Evaluate the good work between M4 and 64 bits architecture

Mini-gmp.h

int mpz_cmp_d (const mpz_t, **double**);

int mpz_cmpabs_d (const mpz_t, **double**);

double mpz_get_d (const mpz_t);

void mpz_set_d (mpz_t, **double**);

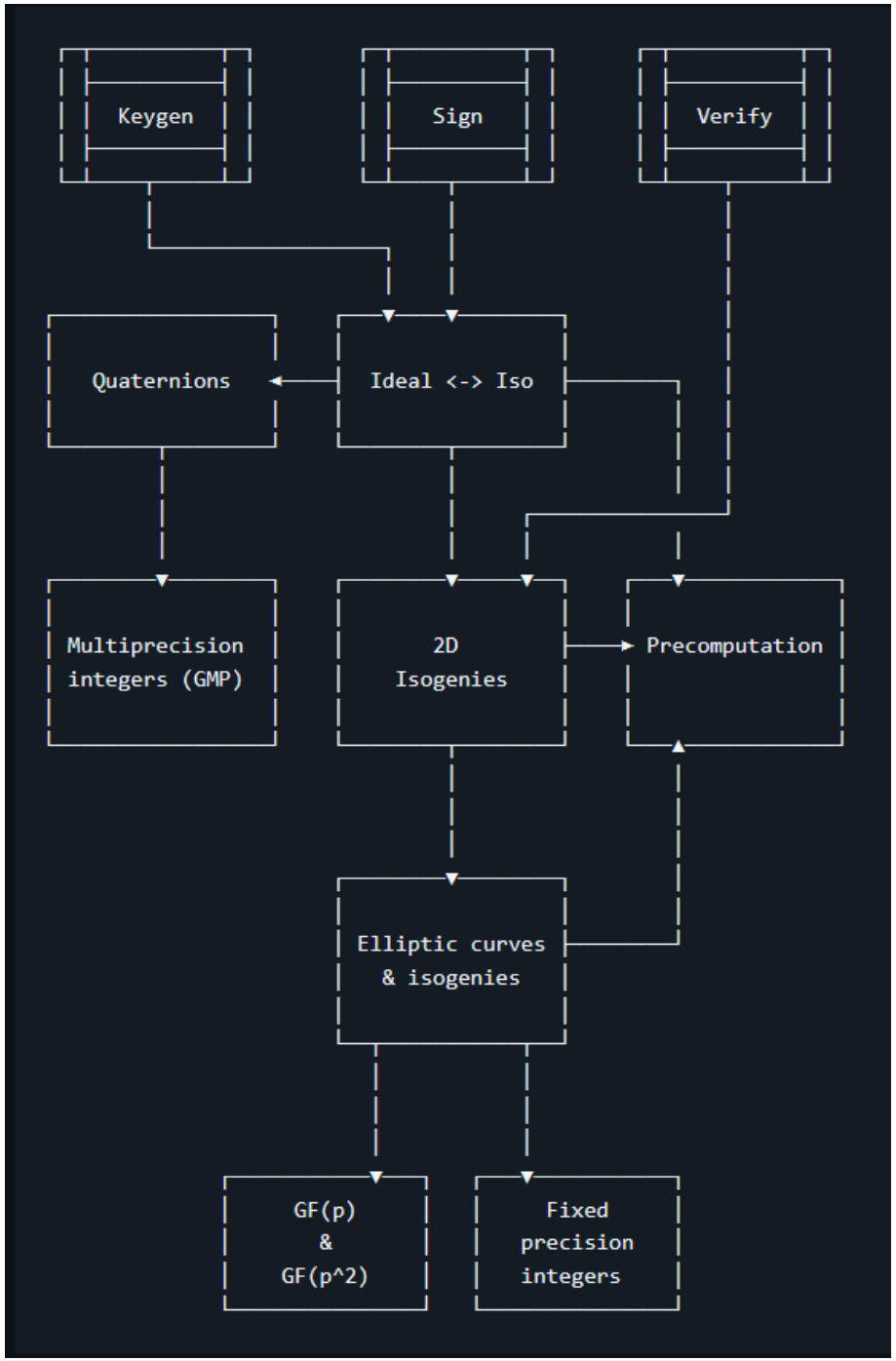
void mpz_init_set_d (mpz_t, **double**);

Mini-gmp-extra.h

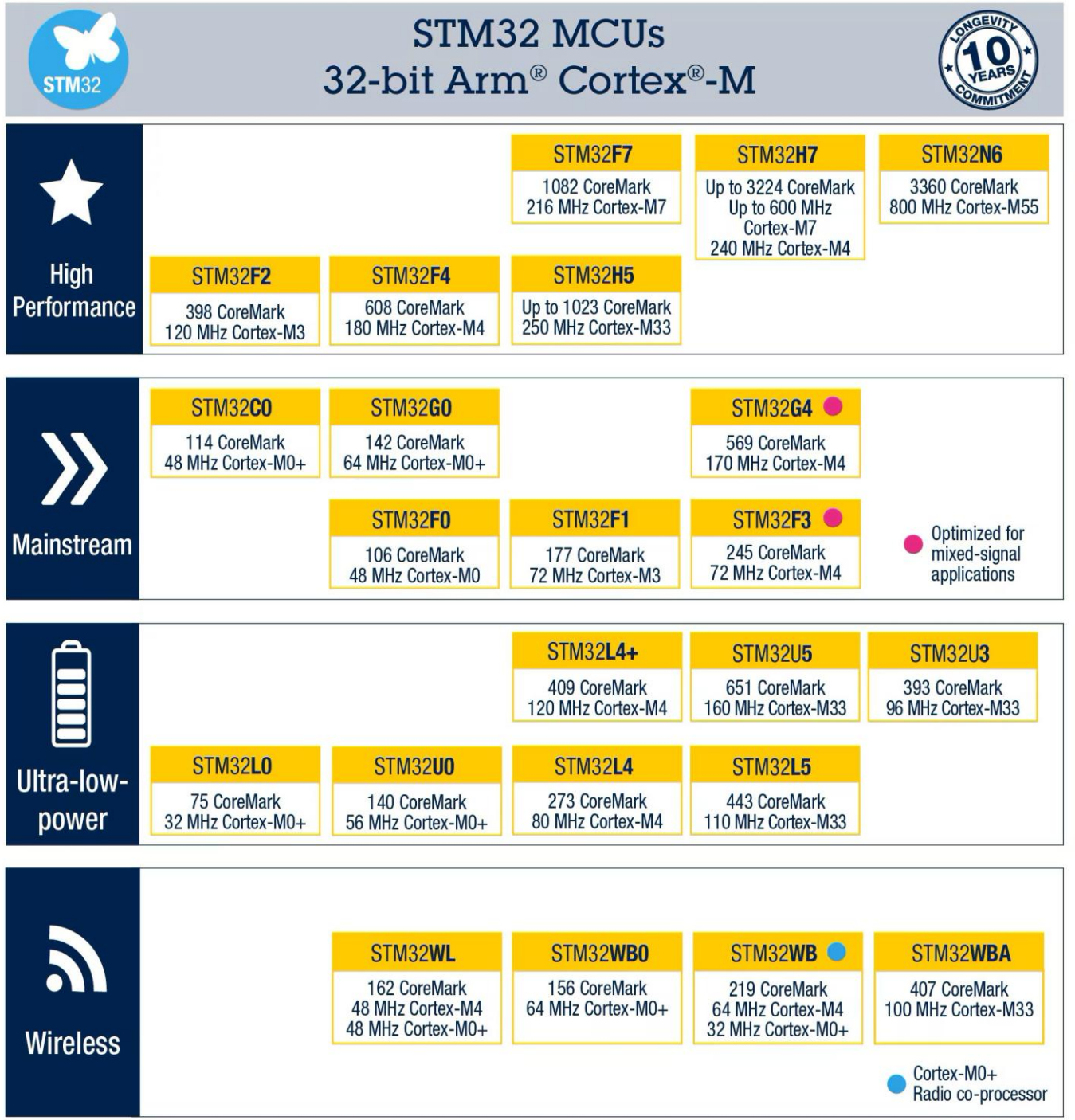
double mpz_get_d_2exp(signed long int *exp, const mpz_t op);

double mini_mpz_get_d_2exp(signed long int *exp, const mpz_t op);

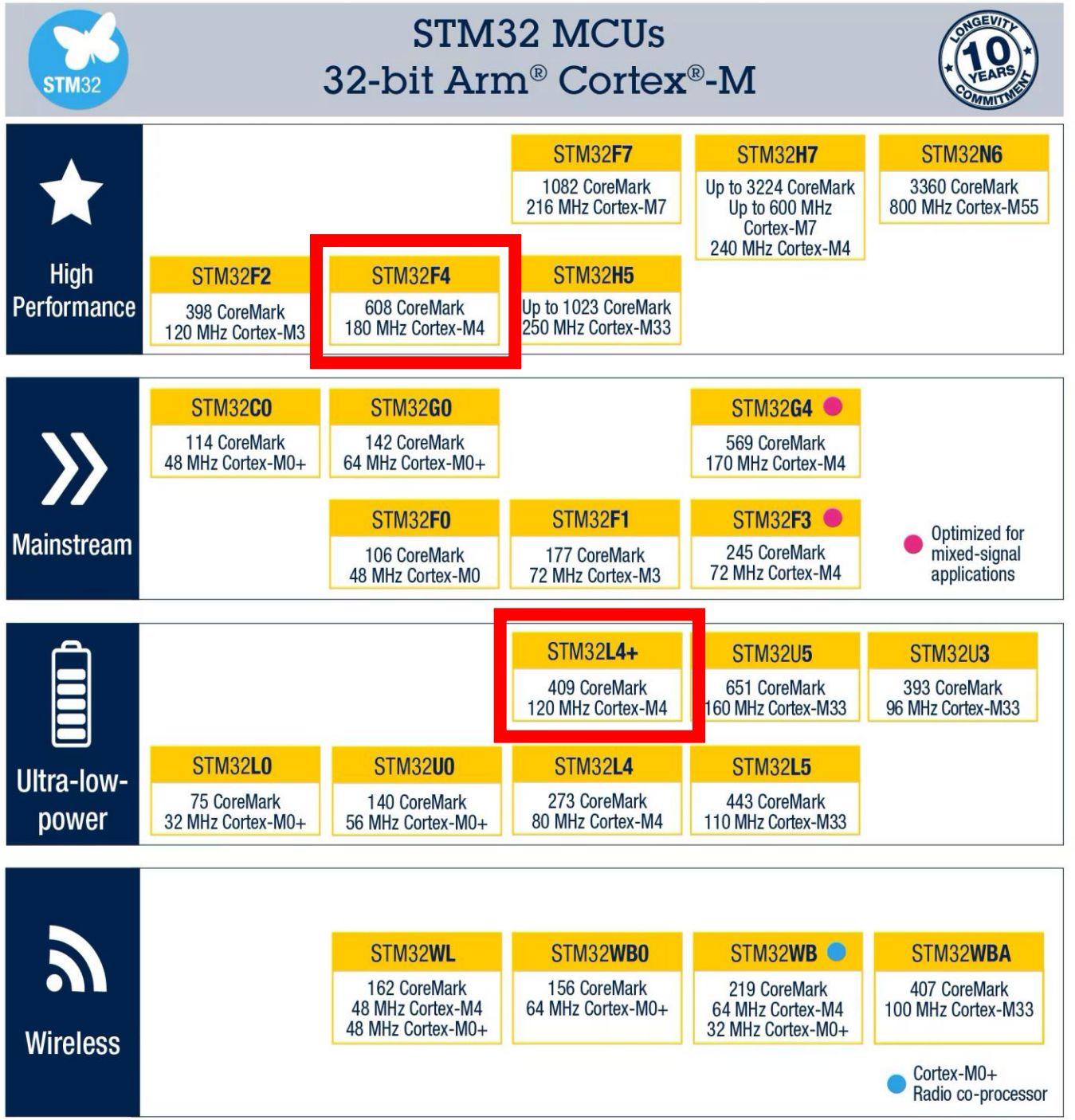
SQIsign Dependencies



https://github.com/SQISign/the-sqisign



https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html?ecmp=tt21056_gl_link_may2021



https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html?ecmp=tt21056_gl_link_may2021