SiliconLabs / gecko_sdk Public

Code Issues 16 Pull requests 3 Actions Projects Security ···

gecko_sdk / platform / bootloader / core / btl_bootload.c

Newer Older

100644 | 980 lines (852 sloc) | 31.8 KB

Gecko SDK 4.0.0 12 months ago

```
/*************************
 * @file
 * @brief Bootloading func
 *************************
 * # License
 * <b>Copyright 2021 Silic
 *************************
 *
 * The licensor of this sc
 * software is governed by
 * Agreement (MSLA) availa
 * www.silabs.com/about-us
 * software is distributed
 * sections of the MSLA ap
 *
 *************************
#include "config/btl_confi

#include "btl_bootload.h"
#include "btl_reset.h"
#include "btl_util.h"

#if defined(SEMAILBOX_PRES
MISRAC_DISABLE
#include "em_se.h"
MISRAC_ENABLE
#endif

// Interface
```

```
#include "api/btl_interfac
#include "api/application_

// Image parser
#include "parser/gbl/btl_g

// Security algorithms
#include "security/btl_sec
#include "security/btl_sec
#include "security/btl_crc
#include "security/btl_sec

// Flashing
#include "core/flash/btl_i

// Debug
#include "debug/btl_debug.

// Get memcpy
#include <string.h>

#ifdef __ICCARM__
// Silence MISRA warning c
#pragma diag_suppress=Pm04
```

Gecko SDK 4.0.2 9 months ago

```
// Silence MISRA warning c
#pragma diag_suppress=Pm02
```

Gecko SDK 4.0.0 12 months ago

```
#endif

//
// Option validation
//
#if defined(BOOTLOADER_ROL
#if defined(_SILICON_LABS_
#error "Rollback protectio
#endif

#endif // defined(BOOTLOAD

#if defined(BOOTLOADER_SUP
#if !defined(_SILICON_LABS
#error "Certificate not su
#endif
#endif // defined(BOOTLOAD

// -----------------------
// Local type declarations
static bool bootload_verif

static void flashData(uint
                      uint
                      size

static bool getSignatureX(


#if defined(BOOTLOADER_ROL
static bool checkResetMagi
static bool checkMaxVersio
static uint32_t getHighest
#endif

// -----------------------
// Defines

#if defined(BOOTLOADER_ROL
#define SL_GBL_APPLICATION
#define SL_GBL_APPLICATION
#define SL_GBL_APPLICATION
#define SL_GBL_UINT32_MAX_
#endif

// -----------------------
// Local functions


#if defined(BOOTLOADER_ROL
```

```c
static bool checkMaxVersic
{
  uint32_t *versionMaxMagi
  if (*versionMaxMagicPtr
    return true;
  }
  return false;
}

static bool checkResetMagi
{
  uint32_t *versionResetMa
  if (*versionResetMagicPt
    return true;
  }
  return false;
}

static uint32_t getHighest
{
  uint32_t *appVersionStor
  if (checkMaxVersionMagic
    return SL_GBL_UINT32_M
  }

  for (uint32_t i = 0UL; i
    ++appVersionStoragePtr
    if (*appVersionStorage
      return *appVersionSt
    }
  }

  return PARSER_APPLICATIO
}
#endif

static void flashData(uint
                      uint
                      size
{
  const uint32_t pageSize 

  // Erase the page if wri
  if (address % pageSize =
    flash_erasePage(addres
  }

  // Erase all pages that 
  for (uint32_t pageAddres
       pageAddress < (addr
       pageAddress += page
    flash_erasePage(pageAd
  }

  BTL_DEBUG_PRINT("F ");
  BTL_DEBUG_PRINT_WORD_HEX
  BTL_DEBUG_PRINT(" to ");
  BTL_DEBUG_PRINT_WORD_HEX
  BTL_DEBUG_PRINT_LF();

  flash_writeBuffer_dma(ad
}

static bool getSignatureX(
{
  // Check if app properti
  if (bootload_checkApplic
    if (appProperties->sig
      // Application signa
      BTL_DEBUG_PRINTLN("W
      return false;
    }

    // Compatibility check
    if (!bootload_checkApp
      return false;
    }
    *appSignatureX = appPr
  } else {
    *appSignatureX = (uint
  }
  return true;
}

static bool bootload_verif
{
  volatile int32_t retVal
  Sha256Context_t shaState

  BareBootTable_t *appStar
  uint32_t appProps = (uin
  uint32_t appSignatureX,
  ApplicationProperties_t
    (ApplicationProperties

  if (!bootload_checkAppli
    return false;
  }
  if (!bootload_checkAppli
    return false;
  }

#if !defined(_SILICON_LABS
  if (PARSER_REQUIRE_ANTI_
    if (!bootload_verifyAp
      return false;
    }
  }
#endif


#if defined(_SILICON_LABS_
  // Access word 13 to rea
  ApplicationProperties_t
    (ApplicationProperties
  if (!bootload_checkAppli
    return false;
  }
#if !defined(MAIN_BOOTLOAD
  if ((uint32_t)blProperti
    // Make sure that this
    return false;
  }
#endif

  bool gotCert = false;
  if (!bootload_verifyAppl
    return false;
  }
#endif

  if (!getSignatureX(appPr
    return false;
  }

  // Check that signature
  if ((appSignatureX < (ui
      || (appSignatureX <
      || (appSignatureX >
    BTL_DEBUG_PRINTLN("No
    return false;
  }

  // SHA-256 of the entire
  btl_initSha256(&shaState
  btl_updateSha256(&shaSta
                  (const
                  appSign

  btl_finalizeSha256(&shaS

  appSignatureY = appSigna
#if defined(_SILICON_LABS_
  if (PARSER_REQUIRE_CERTI
    if (gotCert) {
      // Application certi
      // Authenticate the
      retVal = btl_verifyE




    } else {
      // Application is di
      // Authenticate the
      retVal = btl_verifyE




    }
  } else {
    // Use "lock bits" key
    retVal = btl_verifyEcd




  }
#else
  retVal = btl_verifyEcdsa





#endif

  if (retVal == BOOTLOADER
    return true;
  } else {
    BTL_DEBUG_PRINTLN("Inv
    return false;
  }
}

// -----------------------
// Global functions

// Callbacks
void bootload_application(



{
  (void) context;
  // Check if addresses to
  if ((address < (uint32_t
      || ((address + lengt
          > (uint32_t)(mai
    BTL_DEBUG_PRINT("OOB 0
    BTL_DEBUG_PRINT_WORD_H
    BTL_DEBUG_PRINT_LF();
    return;
  }

  flashData(address, data,
}

void bootload_bootloaderCa




{
  (void) context;


#if defined(BOOTLOADER_HAS
  if (firstBootloaderTable
    // No first stage pres
    return;
  }
#endif

```

Gecko SDK 4.0.2 9 months ago

```
  // Do not allow overwrit
  // the "lock bits" page.
#if defined(LOCKBITS_BASE)
  && (LOCKBITS_BASE != (FL
  const uint32_t max_addre
#else
  const uint32_t max_addre
#endif
  volatile uint32_t addres

  // OOB checks
  // i) if NOT (BTL_UPGRAD
  //    with integer overf
  if ((offset > (uint32_t)
      || (address >= max_a
    BTL_DEBUG_PRINT("OOB,
    BTL_DEBUG_PRINT_WORD_H
    BTL_DEBUG_PRINT_LF();
    return;
  }
  // ii) Semantically equi
  //     but without the r
  if (length > (uint32_t)
    BTL_DEBUG_PRINT("OOB,
```

Gecko SDK 4.0.0 12 months ago

```
    BTL_DEBUG_PRINT_WORD_H
```

Gecko SDK 4.0.2 9 months ago

```
    BTL_DEBUG_PRINT(", (le
    BTL_DEBUG_PRINT_WORD_H
```

Gecko SDK 4.0.0 12 months ago

```
    BTL_DEBUG_PRINT_LF();
    return;
  }

  // Erase first page of a
  // if the bootloader upg
  // This ensures that app
  // bootloader upgrade ha
  if (offset == 0UL) {
    if (BTL_UPGRADE_LOCATI
      flash_erasePage((uin
    }
  }

  flashData(address, data,
}

bool bootload_checkApplica
{
  if (appProperties == NUL
    return false;
  }

#if (FLASH_BASE > 0x0UL)
  if ((uint32_t)appPropert
    return false;
  }
#endif

  uint8_t magicRev[16U] =
  uint8_t *magic = (uint8_

  for (size_t i = 0U; i <
    if (magicRev[15U - i]
      return false;
    }
  }

  return true;

}

```

```
bool bootload_checkApplica
{
  ApplicationProperties_t
  // Compatibility check c
  if (((appProp->structVer
       >> APPLICATION_PROP
      > (uint32_t)APPLICAT
    return false;
  }
  return true;
}

bool bootload_verifyApplic
{
  BareBootTable_t *appStar
  uint32_t appSp = (uint32
  uint32_t appPc = (uint32
  uint32_t appProps = (uin

  // Check that SP points
  if ((appSp < SRAM_BASE)
    BTL_DEBUG_PRINTLN("SP
    return false;
  }

  // Check that PC points
  if ((appPc < (uint32_t)m
      || (appPc > (FLASH_B
    BTL_DEBUG_PRINTLN("PC
    return false;
  }

  ApplicationProperties_t
    (ApplicationProperties


  // Application propertie
  //
  // 0xFFFFFFFF - Likely u
  // [FLASH_BASE, FLASH_SI
  //   - Pointer to Reset_
  //   - Pointer to Applic
  //   - Pointer to ECDSA 

  if ((appProps < ((uint32
      || (appProps > (FLAS
    // Application propert
    if (BOOTLOADER_ENFORCE
      // Secure boot is en
      // pointer to the si
      // is not valid for 
      BTL_DEBUG_PRINTLN("A
      return false;
    } else {
      // Secure boot is no
      BTL_DEBUG_PRINTLN("N
      return true;
    }
  } else if (BOOTLOADER_EN
    // Secure boot is enfo
    BTL_DEBUG_PRINTLN("Sec
    return bootload_verify
  } else if (bootload_chec
    if (!bootload_checkApp
      return false;
    }
    // Application propert
    // based on signature 
    if (appProperties->sig
      // No signature, app
      BTL_DEBUG_PRINTLN("N
      return true;
    } else if (appProperti

#ifdef BTL_LIB_NO_SUPPORT_
      // Don't support CRC
      BTL_DEBUG_PRINTLN("(
      return true;
#else
      uint32_t crc = btl_c
        (void *)startAddre
        appProperties->sig
        BTL_CRC32_START);
      if (crc == BTL_CRC32
        BTL_DEBUG_PRINTLN(
        return true;
      } else {
        return false;
      }
#endif
    } else {
      // Default to secure
      BTL_DEBUG_PRINTLN("S
      return bootload_veri
    }
  } else {
    // Application propert
    // an application prop
    // Secure boot is not
    // pointer to the Rese
    BTL_DEBUG_PRINTLN("No
    return true;
  }
}

uint32_t bootload_getAppli
{
#if defined(BOOTLOADER_ROL
  return SL_GBL_APPLICATIO
#else
  return 0UL;

#endif
}
```

```

uint32_t* bootload_getAppl
{
#if defined(BOOTLOADER_ROL
  uint32_t endOfBLpage = E
  uint32_t *appVersionStor
  return appVersionStorage
#else
  (void)index;
  return NULL;
#endif
}

bool bootload_storeApplica
{
#if defined(BOOTLOADER_ROL
  BareBootTable_t *appStar
  ApplicationProperties_t
  uint32_t appVersion = ap
  uint32_t emptySlots = bo
  uint32_t highestVersionS
  uint32_t *appVersionStor

  if (!bootload_checkAppli
    return false;
  }
  if (!bootload_checkAppli
    return false;
  }

  if (checkMaxVersionMagic
    // The highest allowed
    // so we do not need t
    return true;
  }

  if (*appVersionStoragePt
    return false;
  }
  if (highestVersionSeen =
    // Do not need to stor
    return true;
  }

  if (appVersion == SL_GBL
    appVersion = SL_GBL_AP
    // Return true eventho
    (void)flash_writeBuffe
    return true;
  }

  // The application that
  // However, this versior
  // downgrade later. This
  // Unless the slots are
  if (emptySlots == 0UL) {
    return false;
  }

  appVersionStoragePtr = b
  (void)flash_writeBuffer_
  return true;
#else
  (void)startAddress;
  return false;
#endif
}

bool bootload_verifyApplic
{
#if defined(BOOTLOADER_ROL
  uint32_t highestVersionS


  // Check for the minimum
  if (PARSER_APPLICATION_M
    return false;
  }
  if (highestVersionSeen >
    return false;
  }

  // Application version i
  // Check if we have empt
  if ((appVersion > highes
    // The new applicatior
    if (bootload_remaining
      return false;
    }
  }

  return true;
#else
  (void)appVersion;
  (void)checkRemainingAppl
  return false;
#endif
}

uint32_t bootload_remainin
{
#if defined(BOOTLOADER_ROL
  uint32_t *appVersionStor
  if (checkMaxVersionMagic
    return 0UL;
  }

  for (uint32_t i = 0UL; i
    appVersionStoragePtr =
    if (*appVersionStorage
      return (SL_GBL_APPLI

    }
  }

  return 0UL;
#else
  return 0UL;
#endif
}

void bootload_storeApplica
{
#if defined(BOOTLOADER_ROL
  uint32_t *appVersionRese
  uint32_t appVersionReset
  (void)flash_writeBuffer_
#else
  return;
#endif
}

void bootload_removeStored
{
#if defined(BOOTLOADER_ROL
  uint32_t *appVersionRese
  if ((bootload_remainingA
      && checkResetMagic()
    // Not empty and reset
    uint32_t versionStorag
    (void)flash_erasePage(
  }
#else
  return;
#endif
}

bool bootload_gotCertifica
{

#if defined(BOOTLOADER_SUP
  if (appProp == NULL) {
    return false;
  }

  ApplicationProperties_t
  // Compatibility check c
  // The application prope
  // does not contain the
  if (((appProperties->str
       >> APPLICATION_PROP
    return false;
  }

  if (((appProperties->str
       >> APPLICATION_PROP
    return false;
  }

  if (appProperties->cert
    return false;
  }

  return true;
#else
  (void)appProp;
  return false;
#endif
}

bool bootload_verifyCertif
{
#if defined(BOOTLOADER_SUP
  if (cert == NULL) {
    return false;
  }
  ApplicationCertificate_t


  volatile int32_t retVal
  Sha256Context_t shaState

  // Access word 13 to rea
  ApplicationProperties_t
    (ApplicationProperties
  if (!bootload_checkAppli
    return false;
  }
#if !defined(MAIN_BOOTLOAD
  if ((uint32_t)blProperti
    // Make sure that this
    return false;
  }
#endif

  // Application cert vers
  // the running bootloade
  if (blProperties->cert->
    return false;
  } else {
    // Check ECDSA signing
    btl_initSha256(&shaSta
    btl_updateSha256(&shaS
                     (cons
                     72U);
    btl_finalizeSha256(&sh

    // Use the public key
    // to verify the certi
    // has been validated
    retVal = btl_verifyEcd




    if (retVal != BOOTLOAD
      return false;
    }
    return true;
  }
#else
  (void)cert;
  return false;
#endif
}

bool bootload_verifyApplic
{
#if defined(BOOTLOADER_SUP
  ApplicationProperties_t
  bool *gotCertificate = (
  *gotCertificate = bootlo
  if (*gotCertificate) {
    // Validate Cert
    if (!bootload_verifyCe
      // Cert found, but i
      return false;
    }
  }
#if defined(BOOTLOADER_REJ
  else {
    return false;
  }
#endif
  return true;
#else
  (void)appProp;
  (void)gotCert;
  return true;
#endif
}

// -----------------------
// Secure Element functior
```

```

bool bootload_commitBootl
{
  // Check CRC32 checksum
  uint32_t crc = btl_crc32
  if (crc != BTL_CRC32_END
    // CRC32 check failed.
    return false;
  }

#if defined(SEMAILBOX_PRES
#if defined(_CMU_CLKEN1_SE
  CMU->CLKEN1_SET = CMU_CL
#endif

  // Init with != SE_RESPO
  SE_Response_t response =

  // Verify upgrade image
  SE_Command_t checkImage
  SE_addParameter(&checkIm
  SE_addParameter(&checkIm

  SE_executeCommand(&check
  response = SE_readComman

  if (response != SE_RESPO
    return false;
  }
#endif

#if !defined(_SILICON_LABS
  // Set Reset Magic to si
  // Doing this to make su
  // (Those versions will

  bootload_storeApplicatio
 #endif
```

```

#if defined(SEMAILBOX_PRES
  // Set reset code for wh
  reset_setResetReason(BOO

  // Apply upgrade image
  SE_Command_t applyImage
  SE_addParameter(&applyIm
  SE_addParameter(&applyIm

  SE_executeCommand(&apply

  // Should never get here
  response = SE_readComman
  return false;
#elif defined(CRYPTOACC_PR
  // Set reset code for wh
  reset_setResetReason(BOO

  // Apply upgrade image
  SE_Command_t applyImage
  SE_addParameter(&applyIm
  SE_addParameter(&applyIm

  SE_executeCommand(&apply

  // Should never get here
  return false;
#else
  (void) upgradeAddress;
  (void) size;
  // Reboot and apply upgr
  reset_resetWithReason(BO

  // Should never get here
  return false;
#endif
```

```
}

#if defined(_MSC_PAGELOCK6
bool bootload_lockApplicat
{
  if (endAddress == 0U) {
    // It is assumed that
    BareBootTable_t *appSt
    ApplicationProperties_
    bool retVal = getSigna
    if (!retVal) {
      BTL_DEBUG_PRINTLN("W
      return false;
    }
  }

  if (startAddress > endAd
    return false;
  }

  uint32_t volatile * page
  const uint32_t pageSize
  uint32_t pageNo = ((star
  uint32_t endPageNo = ((e

#if defined(CMU_CLKEN1_MSC
  CMU->CLKEN1_SET = CMU_CL
#endif
  while (pageNo < endPageN
    pageLockAddr = (uint32
    // Find the page lock
    pageLockAddr = &pageLo
    *pageLockAddr = (1UL <
    pageNo += 1U;
  }

#if defined(CRYPTOACC_PRES
  CMU->CLKEN1_CLR = CMU_CL
```

```c
#endif
  return true;
}
#endif

#if defined(SEMAILBOX_PRES
bool bootload_checkSeUpgra
{
#if defined(_CMU_CLKEN1_SE
  CMU->CLKEN1_SET = CMU_CL
#endif

  // Init with != SE_RESPO
  SE_Response_t response =
  uint32_t runningVersion

  SE_Command_t getVersion
  SE_DataTransfer_t dataOu
  SE_addDataOutput(&getVer

  SE_executeCommand(&getVe
  response = SE_readComman

  if (response != SE_RESPO
    // Failed to communica
    return false;
  }

  // Only allow upgrade if
  if (runningVersion < upg
    return true;
  } else {
    return false;
  }
}


bool bootload_commitSeUpgr
```

```
{
#if defined(_CMU_CLKEN1_SE
  CMU->CLKEN1_SET = CMU_CL
#endif

  // Init with != SE_RESPO
  SE_Response_t response =

  // Verify upgrade image
  SE_Command_t checkImage
  SE_addParameter(&checkIm

  SE_executeCommand(&check
  response = SE_readComman

  if (response != SE_RESPO
    return false;
  }

  // Set reset code for wh
  reset_setResetReason(BOO

  // Apply upgrade image
  SE_Command_t applyImage
  SE_addParameter(&applyIm

  SE_executeCommand(&apply

  // Should never get here
  response = SE_readComman
  return false;
}

#elif defined(CRYPTOACC_PR
bool bootload_checkSeUpgra

{
  uint32_t runningVersion
  if (SE_getVersion(&runni
    // Failed to communica
    return false;
  }
  // Only allow upgrade if
  if (runningVersion < upg
    return true;
  }
  return false;
}

bool bootload_commitSeUpgr
{
  // Set reset code for wh
  reset_setResetReason(BOO

  // Apply upgrade image
  SE_Command_t applyImage
  SE_addParameter(&applyIm

  SE_executeCommand(&apply

  // Should never get here
  return false;
}
#endif // defined(CRYPTOAC
```