100644 | 980 lines (852 sloc) | 31.8 KB

Gecko SDK 4.0.0 12 months ago

```
1  /*************************
2   * @file
3   * @brief Bootloading func
4   *************************
5   * # License
6   * <b>Copyright 2021 Silic
7   *************************
8   *
9   * The licensor of this so
10  * software is governed by
11  * Agreement (MSLA) availa
12  * www.silabs.com/about-us
13  * software is distributed
14  * sections of the MSLA ap
15  *
16  *************************
17 #include "config/btl_confi
18
19 #include "btl_bootload.h"
20 #include "btl_reset.h"
21 #include "btl_util.h"
22
23 #if defined(SEMAILBOX_PRES
24 MISRAC_DISABLE
25 #include "em_se.h"
26 MISRAC_ENABLE
27 #endif
28
29 // Interface
```

		30	#include "api/btl_interfac
		31	#include "api/application_
		32	
		33	// Image parser
		34	#include "parser/gbl/btl_g
		35	
		36	// Security algorithms
		37	#include "security/btl_sec
		38	#include "security/btl_sec
		39	#include "security/btl_cry
		40	#include "security/btl_sec
		41	
		42	// Flashing
		43	#include "core/flash/btl_i
		44	
		45	// Debug
		46	#include "debug/btl_debug.
		47	
		48	// Get memcpy
		49	#include <string.h>
		50	
		51	#ifdef __ICCARM__
		52	// Silence MISRA warning c
		53	#pragma diag_suppress=Pm04
Gecko SDK 4.0.2	9 months ago	54	// Silence MISRA warning c
		55	#pragma diag_suppress=Pm02
Gecko SDK 4.0.0	12 months ago	56	#endif
		57	
		58	//
		59	// Option validation
		60	//
		61	#if defined(BOOTLOADER_ROL
		62	#if defined(_SILICON_LABS_
		63	#error "Rollback protectio
		64	#endif
		65	#endif // defined(BOOTLOAD
		66	

```
#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
```

```
104 static bool checkMaxVersio
105 {
106   uint32_t *versionMaxMagi
107   if (*versionMaxMagicPtr
108     return true;
109   }
110   return false;
111 }
112
113 static bool checkResetMagi
114 {
115   uint32_t *versionResetMa
116   if (*versionResetMagicPt
117     return true;
118   }
119   return false;
120 }
121
122 static uint32_t getHighest
123 {
124   uint32_t *appVersionStor
125   if (checkMaxVersionMagic
126     return SL_GBL_UINT32_M
127   }
128
129   for (uint32_t i = 0UL; i
130     ++appVersionStoragePtr
131     if (*appVersionStorage
132       return *appVersionSt
133     }
134   }
135
136   return PARSER_APPLICATIO
137 }
138 #endif
139
140 static void flashData(uint
```

```
141                         uint
142                         size
143 {
144   const uint32_t pageSize
145
146   // Erase the page if wri
147   if (address % pageSize =
148     flash_erasePage(addres
149   }
150
151   // Erase all pages that
152   for (uint32_t pageAddres
153         pageAddress < (addr
154         pageAddress += page
155     flash_erasePage(pageAd
156   }
157
158   BTL_DEBUG_PRINT("F ");
159   BTL_DEBUG_PRINT_WORD_HEX
160   BTL_DEBUG_PRINT(" to ");
161   BTL_DEBUG_PRINT_WORD_HEX
162   BTL_DEBUG_PRINT_LF();
163
164   flash_writeBuffer_dma(ad
165 }
166
167 static bool getSignatureX(
168 {
169   // Check if app properti
170   if (bootload_checkApplic
171     if (appProperties->sig
172       // Application signa
173       BTL_DEBUG_PRINTLN("W
174       return false;
175     }
176     // Compatibility check
177     if (!bootload_checkApp
```

```
178         return false;
179       }
180       *appSignatureX = appPr
181     } else {
182       *appSignatureX = (uint
183     }
184     return true;
185   }
186
187   static bool bootload_verif
188   {
189     volatile int32_t retVal
190     Sha256Context_t shaState
191
192     BareBootTable_t *appStar
193     uint32_t appProps = (uin
194     uint32_t appSignatureX,
195     ApplicationProperties_t
196       (ApplicationProperties
197
198     if (!bootload_checkAppli
199       return false;
200     }
201     if (!bootload_checkAppli
202       return false;
203     }
204
205   #if !defined(_SILICON_LABS
206     if (PARSER_REQUIRE_ANTI_
207       if (!bootload_verifyAp
208         return false;
209       }
210     }
211   #endif
212
213   #if defined(_SILICON_LABS_
214     // Access word 13 to rea
```

```
215   ApplicationProperties_t
216     (ApplicationProperties
217   if (!bootload_checkAppli
218     return false;
219   }
220 #if !defined(MAIN_BOOTLOAD
221   if ((uint32_t)blProperti
222     // Make sure that this
223     return false;
224   }
225 #endif
226
227   bool gotCert = false;
228   if (!bootload_verifyAppl
229     return false;
230   }
231 #endif
232
233   if (!getSignatureX(appPr
234     return false;
235   }
236
237   // Check that signature
238   if ((appSignatureX < (ui
239       || (appSignatureX <
240       || (appSignatureX >
241     BTL_DEBUG_PRINTLN("No
242     return false;
243   }
244
245   // SHA-256 of the entire
246   btl_initSha256(&shaState
247   btl_updateSha256(&shaSta
248                    (const
249                    appSigr
250   btl_finalizeSha256(&shaS
251
```

```
252     appSignatureY = appSigna
253   #if defined(_SILICON_LABS_
254     if (PARSER_REQUIRE_CERTI
255       if (gotCert) {
256         // Application certi
257         // Authenticate the
258         retVal = btl_verifyE
259
260
261
262
263       } else {
264         // Application is di
265         // Authenticate the
266         retVal = btl_verifyE
267
268
269
270
271       }
272     } else {
273       // Use "lock bits" key
274       retVal = btl_verifyEcd
275
276
277
278
279     }
280   #else
281     retVal = btl_verifyEcdsa
282
283
284
285
286   #endif

287     if (retVal == BOOTLOADER
288       return true;
```

```
289     } else {
290       BTL_DEBUG_PRINTLN("Inv
291       return false;
292     }
293   }
294
295   // ----------------------
296   // Global functions
297
298   // Callbacks
299   void bootload_application(
300
301
302
303   {
304     (void) context;
305     // Check if addresses to
306     if ((address < (uint32_t
307         || ((address + lengt
308             > (uint32_t)(mai
309       BTL_DEBUG_PRINT("OOB 0
310       BTL_DEBUG_PRINT_WORD_H
311       BTL_DEBUG_PRINT_LF();
312       return;
313     }
314
315     flashData(address, data,
316   }
317
318   void bootload_bootloaderCa
319
320
321
322   {
323     (void) context;
324
325   #if defined(BOOTLOADER_HAS
```

		326	`  if (firstBootloaderTable`
		327	`    // No first stage pres`
		328	`    return;`
		329	`  }`
		330	`#endif`
		331	
Gecko SDK 4.0.2	9 months ago	332	`  // Do not allow overwrit`
		333	`  // the "lock bits" page.`
		334	`#if defined(LOCKBITS_BASE)`
		335	`  && (LOCKBITS_BASE != (FL`
		336	`  const uint32_t max_addre`
		337	`#else`
		338	`  const uint32_t max_addre`
		339	`#endif`
		340	`  volatile uint32_t addres`
		341	
		342	`  // OOB checks`
		343	`  // i) if NOT (BTL_UPGRAD`
		344	`  //    with integer overf`
		345	`  if ((offset > (uint32_t)`
		346	`      || (address >= max_a`
		347	`    BTL_DEBUG_PRINT("OOB,`
		348	`    BTL_DEBUG_PRINT_WORD_H`
		349	`    BTL_DEBUG_PRINT_LF();`
		350	`    return;`
		351	`  }`
		352	`  // ii) Semantically equi`
		353	`  //     but without the r`
		354	`  if (length > (uint32_t)`
		355	`    BTL_DEBUG_PRINT("OOB,`
Gecko SDK 4.0.0	12 months ago	356	`    BTL_DEBUG_PRINT_WORD_H`
Gecko SDK 4.0.2	9 months ago	357	`    BTL_DEBUG_PRINT(", (le`
		358	`    BTL_DEBUG_PRINT_WORD_H`
Gecko SDK 4.0.0	12 months ago	359	`    BTL_DEBUG_PRINT_LF();`
		360	`    return;`
		361	`  }`
		362	

```
363   // Erase first page of a
364   // if the bootloader upg
365   // This ensures that app
366   // bootloader upgrade ha
367   if (offset == 0UL) {
368     if (BTL_UPGRADE_LOCATI
369       flash_erasePage((uir
370     }
371   }
372
373   flashData(address, data,
374 }
375
376 bool bootload_checkApplica
377 {
378   if (appProperties == NUL
379     return false;
380   }
381
382 #if (FLASH_BASE > 0x0UL)
383   if ((uint32_t)appPropert
384     return false;
385   }
386 #endif
387
388   uint8_t magicRev[16U] =
389   uint8_t *magic = (uint8_
390
391   for (size_t i = 0U; i <
392     if (magicRev[15U - i]
393       return false;
394     }
395   }
396
397   return true;
398 }
399
```

```
400  bool bootload_checkApplica
401  {
402    ApplicationProperties_t
403    // Compatibility check c
404    if (((appProp->structVer
405         >> APPLICATION_PROP
406        > (uint32_t)APPLICAT
407      return false;
408    }
409    return true;
410  }
411
412  bool bootload_verifyApplic
413  {
414    BareBootTable_t *appStar
415    uint32_t appSp = (uint32
416    uint32_t appPc = (uint32
417    uint32_t appProps = (uin
418
419    // Check that SP points
420    if ((appSp < SRAM_BASE)
421      BTL_DEBUG_PRINTLN("SP
422      return false;
423    }
424
425    // Check that PC points
426    if ((appPc < (uint32_t)n
427        || (appPc > (FLASH_E
428      BTL_DEBUG_PRINTLN("PC
429      return false;
430    }
431
432    ApplicationProperties_t
433      (ApplicationProperties
434
435    // Application propertie
436    //
```

```
437     // 0xFFFFFFFF - Likely u
438     // [FLASH_BASE, FLASH_SI
439     //   - Pointer to Reset_
440     //   - Pointer to Applic
441     //   - Pointer to ECDSA
442
443     if ((appProps < ((uint32
444         || (appProps > (FLAS
445       // Application propert
446       if (BOOTLOADER_ENFORCE
447         // Secure boot is en
448         // pointer to the si
449         // is not valid for
450         BTL_DEBUG_PRINTLN("A
451         return false;
452       } else {
453         // Secure boot is no
454         BTL_DEBUG_PRINTLN("N
455         return true;
456       }
457     } else if (BOOTLOADER_EN
458       // Secure boot is enfo
459       BTL_DEBUG_PRINTLN("Sec
460       return bootload_verify
461     } else if (bootload_chec
462       if (!bootload_checkApp
463         return false;
464       }
465       // Application propert
466       // based on signature
467       if (appProperties->sig
468         // No signature, app
469         BTL_DEBUG_PRINTLN("N
470         return true;
471       } else if (appProperti

472 #ifdef BTL_LIB_NO_SUPPORT_
473         // Don't support CRC
```

```
474         BTL_DEBUG_PRINTLN("C
475         return true;
476 #else
477         uint32_t crc = btl_c
478           (void *)startAddre
479           appProperties->sig
480           BTL_CRC32_START);
481         if (crc == BTL_CRC32
482           BTL_DEBUG_PRINTLN(
483           return true;
484         } else {
485           return false;
486         }
487 #endif
488       } else {
489         // Default to secure
490         BTL_DEBUG_PRINTLN("S
491         return bootload_veri
492       }
493     } else {
494       // Application propert
495       // an application prop
496       // Secure boot is not
497       // pointer to the Rese
498       BTL_DEBUG_PRINTLN("No
499       return true;
500     }
501 }
502
503 uint32_t bootload_getAppli
504 {
505 #if defined(BOOTLOADER_ROL
506   return SL_GBL_APPLICATIO
507 #else
508   return 0UL;
509 #endif
510 }
```

```
511
512 uint32_t* bootload_getAppl
513 {
514 #if defined(BOOTLOADER_ROL
515   uint32_t endOfBLpage = E
516   uint32_t *appVersionStor
517   return appVersionStorage
518 #else
519   (void)index;
520   return NULL;
521 #endif
522 }
523
524 bool bootload_storeApplica
525 {
526 #if defined(BOOTLOADER_ROL
527   BareBootTable_t *appStar
528   ApplicationProperties_t
529   uint32_t appVersion = ap
530   uint32_t emptySlots = bo
531   uint32_t highestVersionS
532   uint32_t *appVersionStor
533
534   if (!bootload_checkAppli
535     return false;
536   }
537   if (!bootload_checkAppli
538     return false;
539   }
540
541   if (checkMaxVersionMagic
542     // The highest allowed
543     // so we do not need t
544     return true;
545   }
546   if (*appVersionStoragePt
547     return false;
```

```
548     }
549     if (highestVersionSeen =
550       // Do not need to stor
551       return true;
552     }
553
554     if (appVersion == SL_GBL
555       appVersion = SL_GBL_AP
556       // Return true eventho
557       (void)flash_writeBuffe
558       return true;
559     }
560
561     // The application that
562     // However, this versior
563     // downgrade later. This
564     // Unless the slots are
565     if (emptySlots == 0UL) {
566       return false;
567     }
568
569     appVersionStoragePtr = b
570     (void)flash_writeBuffer_
571     return true;
572   #else
573     (void)startAddress;
574     return false;
575   #endif
576   }
577
578   bool bootload_verifyApplic
579   {
580   #if defined(BOOTLOADER_ROL
581     uint32_t highestVersionS
582
583     // Check for the minimum
584     if (PARSER_APPLICATION_M
```

```
585       return false;
586     }
587     if (highestVersionSeen >
588       return false;
589     }
590
591     // Application version i
592     // Check if we have empt
593     if ((appVersion > highes
594       // The new applicatior
595       if (bootload_remaining
596         return false;
597       }
598     }
599
600     return true;
601   #else
602     (void)appVersion;
603     (void)checkRemainingAppU
604     return false;
605   #endif
606   }
607
608   uint32_t bootload_remainir
609   {
610   #if defined(BOOTLOADER_ROL
611     uint32_t *appVersionStor
612     if (checkMaxVersionMagic
613       return 0UL;
614     }
615
616     for (uint32_t i = 0UL; i
617       appVersionStoragePtr =
618       if (*appVersionStorage
619         return (SL_GBL_APPLI

620       }
621     }
```

```
622
623     return 0UL;
624   #else
625     return 0UL;
626   #endif
627   }
628
629   void bootload_storeApplica
630   {
631   #if defined(BOOTLOADER_ROL
632     uint32_t *appVersionRese
633     uint32_t appVersionReset
634     (void)flash_writeBuffer_
635   #else
636     return;
637   #endif
638   }
639
640   void bootload_removeStored
641   {
642   #if defined(BOOTLOADER_ROL
643     uint32_t *appVersionRese
644     if ((bootload_remainingA
645         && checkResetMagic()
646       // Not empty and reset
647       uint32_t versionStorag
648       (void)flash_erasePage(
649     }
650   #else
651     return;
652   #endif
653   }
654
655   bool bootload_gotCertifica
656   {
657   #if defined(BOOTLOADER_SUP
658     if (appProp == NULL) {
```

```
659       return false;
660     }
661
662     ApplicationProperties_t
663     // Compatibility check c
664     // The application prope
665     // does not contain the
666     if (((appProperties->str
667           >> APPLICATION_PROP
668       return false;
669     }
670
671     if (((appProperties->str
672           >> APPLICATION_PROP
673       return false;
674     }
675
676     if (appProperties->cert
677       return false;
678     }
679
680     return true;
681   #else
682     (void)appProp;
683     return false;
684   #endif
685   }
686
687   bool bootload_verifyCertif
688   {
689   #if defined(BOOTLOADER_SUP
690     if (cert == NULL) {
691       return false;
692     }
693     ApplicationCertificate_t
694
695     volatile int32_t retVal
```

```
696     Sha256Context_t shaState
697
698     // Access word 13 to rea
699     ApplicationProperties_t
700       (ApplicationProperties
701     if (!bootload_checkAppli
702       return false;
703     }
704   #if !defined(MAIN_BOOTLOAD
705     if ((uint32_t)blProperti
706       // Make sure that this
707       return false;
708     }
709   #endif
710
711     // Application cert vers
712     // the running bootloade
713     if (blProperties->cert->
714       return false;
715     } else {
716       // Check ECDSA signing
717       btl_initSha256(&shaSta
718       btl_updateSha256(&shaS
719                        (cons
720                        72U);
721       btl_finalizeSha256(&sh
722
723       // Use the public key
724       // to verify the certi
725       // has been validated
726       retVal = btl_verifyEcd
727
728
729
730
731       if (retVal != BOOTLOAD
732         return false;
```

```
733       }
734       return true;
735     }
736   #else
737     (void)cert;
738     return false;
739   #endif
740   }
741
742   bool bootload_verifyApplic
743   {
744   #if defined(BOOTLOADER_SUP
745     ApplicationProperties_t
746     bool *gotCertificate = (
747     *gotCertificate = bootlo
748     if (*gotCertificate) {
749       // Validate Cert
750       if (!bootload_verifyCe
751         // Cert found, but i
752         return false;
753       }
754     }
755   #if defined(BOOTLOADER_REJ
756     else {
757       return false;
758     }
759   #endif
760     return true;
761   #else
762     (void)appProp;
763     (void)gotCert;
764     return true;
765   #endif
766   }
767
768   // ----------------------
769   // Secure Element functior
```

```
770
771 bool bootload_commitBootlo
772 {
773   // Check CRC32 checksum
774   uint32_t crc = btl_crc32
775   if (crc != BTL_CRC32_END
776     // CRC32 check failed.
777     return false;
778   }
779
780 #if defined(SEMAILBOX_PRES
781 #if defined(_CMU_CLKEN1_SE
782   CMU->CLKEN1_SET = CMU_CL
783 #endif
784
785   // Init with != SE_RESPO
786   SE_Response_t response =
787
788   // Verify upgrade image
789   SE_Command_t checkImage
790   SE_addParameter(&checkIm
791   SE_addParameter(&checkIm
792
793   SE_executeCommand(&check
794   response = SE_readCommar
795
796   if (response != SE_RESPO
797     return false;
798   }
799 #endif
800
801 #if !defined(_SILICON_LABS
802   // Set Reset Magic to si
803   // Doing this to make su
804   // (Those versions will
805   bootload_storeApplicatio
806  #endif
```

```
807
808 #if defined(SEMAILBOX_PRES
809   // Set reset code for wh
810   reset_setResetReason(BOO
811
812   // Apply upgrade image
813   SE_Command_t applyImage
814   SE_addParameter(&applyIm
815   SE_addParameter(&applyIm
816
817   SE_executeCommand(&apply
818
819   // Should never get here
820   response = SE_readCommar
821   return false;
822 #elif defined(CRYPTOACC_PR
823   // Set reset code for wh
824   reset_setResetReason(BOO
825
826   // Apply upgrade image
827   SE_Command_t applyImage
828   SE_addParameter(&applyIm
829   SE_addParameter(&applyIm
830
831   SE_executeCommand(&apply
832
833   // Should never get here
834   return false;
835 #else
836   (void) upgradeAddress;
837   (void) size;
838   // Reboot and apply upgr
839   reset_resetWithReason(BO
840
841   // Should never get here

842   return false;
843 #endif
```

```
844  }
845
846  #if defined(_MSC_PAGELOCK(
847  bool bootload_lockApplicat
848  {
849    if (endAddress == 0U) {
850      // It is assumed that
851      BareBootTable_t *appSt
852      ApplicationProperties_
853      bool retVal = getSigna
854      if (!retVal) {
855        BTL_DEBUG_PRINTLN("V
856        return false;
857      }
858    }
859
860    if (startAddress > endAc
861      return false;
862    }
863
864    uint32_t volatile * page
865    const uint32_t pageSize
866    uint32_t pageNo = ((star
867    uint32_t endPageNo = ((e
868
869  #if defined(CMU_CLKEN1_MSC
870    CMU->CLKEN1_SET = CMU_CL
871  #endif
872    while (pageNo < endPageN
873      pageLockAddr = (uint32
874      // Find the page lock
875      pageLockAddr = &pageLo
876      *pageLockAddr = (1UL <
877      pageNo += 1U;
878    }

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

```
#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_readCommar

  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_readCommar

  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_readCommar
  return false;
}

#elif defined(CRYPTOACC_PR
bool bootload_checkSeUpgra

{
  uint32_t runningVersion
```

```
955     if (SE_getVersion(&runni
956       // Failed to communica
957       return false;
958     }
959     // Only allow upgrade if
960     if (runningVersion < upg
961       return true;
962     }
963     return false;
964   }
965
966   bool bootload_commitSeUpgr
967   {
968     // Set reset code for wh
969     reset_setResetReason(BOO
970
971     // Apply upgrade image
972     SE_Command_t applyImage
973     SE_addParameter(&applyIm
974
975     SE_executeCommand(&apply
976
977     // Should never get here
978     return false;
979   }
980   #endif // defined(CRYPTOAC
```