Systèmes Embarqués 2

Journal du TP.08 : Interruptions et exceptions

Temps effectué hors des heures de classe

Nous avons effectué 10h de plus en dehors des quatres périodes mises à disposition pour ce TP.

Synthèse des acquis

Acquis :
- Interfaçage C-Assembleur
- Principe théorique des interruptions

Acquis, mais à exercer encore :
- Gestion des interruptions en C

Feedback

Le passage de la théorie à la pratique a été très difficile sur ce TP. Autant nous avons plus ou moins bien compris le principe des interruptions, autant l'implémentation nous a été assez difficile. Avant que vous nous montriez votre solution complète au tableau, nous ne savions pas par quoi commencer.

Fonctionnement

1. interrupt_init()
   (a) interrupt_init_asm()
      i. Désactive les interrupt
      ii. Définie le début des stacks des modes IRQ, Abort, Undef, Supervisor
      iii. Spécifie l'adresse de la table des vecteurs
      iv. Définie l'adresse de la méthode appelé en cas d'IRQ
      v. Sauve la table des vecteurs
   (b) Initialise la table des vecteurs à 0
   (c) Active les interruptions via interrupt_enable()
2. exception_init()
   (a) Attache un handler d'interruption à chaque vecteur de la table
3. Le système est initialisé, lorsqu'un interruption arrive, le system se retrouve dans la fonction init1_handler qui se charge de lancer la méthode correspondante à cette interruption. Si aucune n'existe, le système est gelé. Si une existe, la méthode exception_handler() est appelé et affiche un message.

Codes

```c
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
#include <stdint.h>
#include "interrupt.h"
#include "exception.h"

int main() {
  printf("\n\n\n");
  printf("HEIA-FR - Embedded Systems 2 Laboratory\n");
  printf("Low Level Interrupt Handling on ARM Cortex-A8\n");
  printf("---------------------------------------------\n");
  printf("Initialization...\n");
  interruptionInitialization();
  exceptionInitialization();
  printf("Initialization done\n");
  printf("---------------------------------------------\n\n");
  printf("Test data abort with a miss aligned access\n");
  long l = 0;
  long* pl = (long*) ((char*) &l + 1);
  *pl = 2;

  printf("\nTest supervisor call instruction / software interrupt\n");
  __asm__("svc #1;");

  printf("\nTest a invalid instruction\n");
  __asm__(".word 0xffffffff;");

  printf("\nTest a prefetch abort. This method will never return...\n");
  __asm__("mov pc,#0x00000000;");

  for(;;);
  return 0;
}
```

Listing 1 – main.c

```c
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
#include <stdint.h>
//#include "interrupt.h"

/**
 * initialization method
 */
extern void exceptionInitialization();

#endif
```

Listing 2 – exception.h

```c
#include "exception.h"
#include "interrupt.h"

/**
 * Called by init1_handler when an interrupt with an unknow vector occurs.
 * Display interrupt vector and interrupt parameter in minicom.
 * If exception is interrupt prefetch, freeze the cpu with an infinite loop.
 */
void exceptionHandler(enum interrupt_vectors vector, void* param) {

  printf("ARM Exception with vector %d and param %s\n", vector, (char*) param);
  if (vector == INT_PREFETCH) {
    for (;;)
      ;                    // infinite loop when prefetch exception
  }
}

void exceptionInitialization() {
  interruptionAttach(INT_UNDEF, exceptionHandler, "undefined instruction");
  interruptionAttach(INT_SVC, exceptionHandler, "software interrupt");
  interruptionAttach(INT_PREFETCH, exceptionHandler, "prefetch abort");
  interruptionAttach(INT_DATA, exceptionHandler, "data abort");
}
```

```c
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
#include <stdint.h>

/**
 * ARM interrupt vectors enumeration
 */
enum interrupt_vectors {
  INT_UNDEF,  ///< undefined instruction
  INT_SVC,    ///< supervisor call (software interrupt)
  INT_PREFETCH,   ///< prefetch abort (instruction prefetch)
  INT_DATA,    ///< data abort (data access)
  INT_IRQ,    ///< hardware interrupt request
  INT_FIQ,    ///< hardware fast interrupt request
  INT_NB_VECTORS
};

/**
 * Prototype of the interrupt handler
 *
 * @param vector interrupt vector
 * @param param parameter specified while attaching the interrupt handler
 */
typedef void (*interruptHandler)(enum interrupt_vectors vector, void* param);

/**
 * Method to initialize low level resources of the microprocessor.
 * At least a 8KiB of memory will be allocated for each interrupt vector
 */
extern void interruptionInitialization();

/**
 * Method to attach an interrupt handler to the interrupt vector table
 *
 * @param vector interrupt vector
 * @param routine interrupt handler to attach to the specified vector
 * @param param parameter to be passed as argument while calling the the
 *              specified interrupt handler
 * @return execution status, 0 if success, -1 if already attached
 */
extern int interruptionAttach(enum interrupt_vectors vector,
    interruptHandler routine, void* param);

/**
 * Method to detach an interrupt handler from the interrupt vector table
 *
 * @param vector interrupt vector
 */
extern void interruptionDetach(enum interrupt_vectors vector);

/**
 * Method to enable interrupt requests
 */
extern void interrupt_enable();

/**
 * Method to disable interrupt requests
 *
 * @return value of cpsr before disabling interrupt requests
 */
extern int interrupt_disable();

#endif
```

Listing 3 – interrupt.h

```c
#include "interrupt.h"

// Method implemented in ASM
extern void ASMInterruptionInitialization(void (*)(enum interrupt_vector));

struct interruptionVector {                                          // Vector table entry
  interruptHandler handler;
  void* param;
};

static struct interruptionVector interruptionVectorTable[INT_NB_VECTORS]; // Vector table

/**
 * Called when an interrupt occurs. If there is an handler for the interruption's vector,
 * call this handler. Else, print an error message et freeze the program.
 */
void interruptionHandler(enum interrupt_vectors vector) {
  if (vector < INT_NB_VECTORS) {
    struct interruptionVector* handler = &interruptionVectorTable[vector];
    if (handler->handler != 0) {
      handler->handler(vector, handler->param);
    } else {
      printf("Error 404 - Interrupt handler for vector %d not found", vector);
      for (;;)
        ;
    }
  } else {
    printf("Black hole for vector %d", vector);
    for (;;)
      ;
  }
}

extern void interruptionInitialization() {
  ASMInterruptionInitialization(&interruptionHandler);
  memset(interruptionVectorTable, 0, sizeof(interruptionVectorTable)); // Fill vector table with 0
  interruptEnable();
}

extern int interruptionAttach(enum interrupt_vectors vector,
    interruptHandler function, void* param) {
  int status = -1;
  if (vector < INT_NB_VECTORS) {
    struct interruptionVector* handler = &interruptionVectorTable[vector];
    if (handler->handler == 0) {                                 // Test if there is already
      handler->handler = function;                               // an handler for this
      handler->param = param;                                    // interrupt vector
      status = 0;
    }
  }
  return status;
}

extern void interruptionDetach(enum interrupt_vectors vector) {
  if (vector < INT_NB_VECTORS) {
    interruptionVectorTable[vector].handler = 0;
  }
}
```

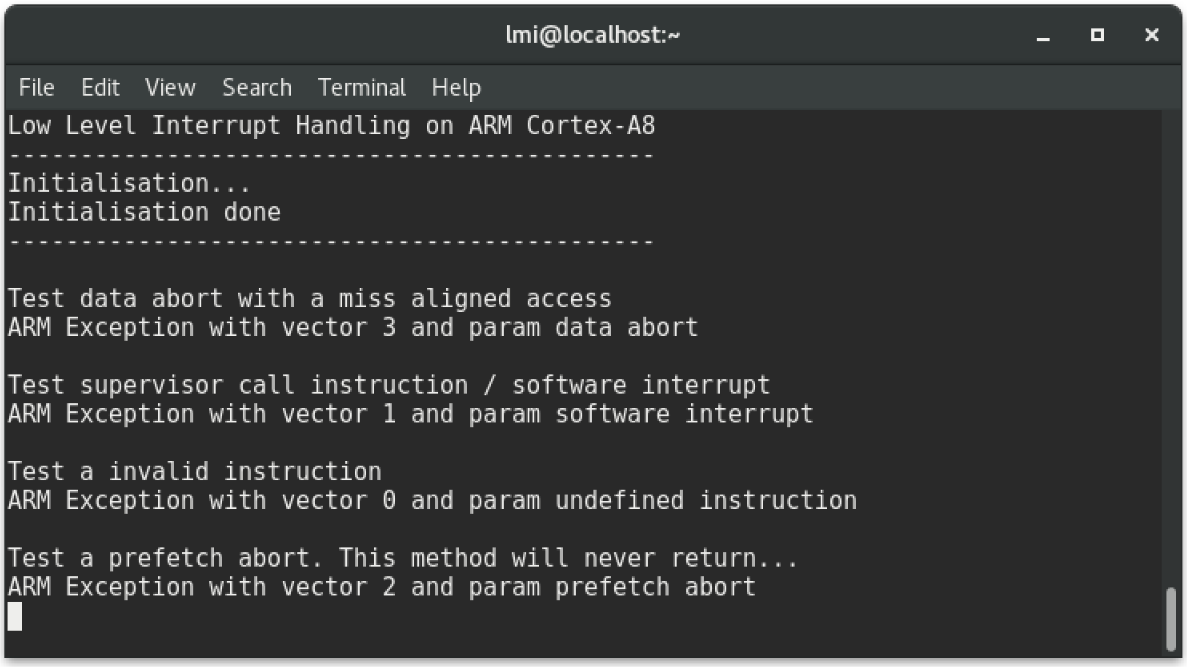
Listing 4 – interrupt.c

```
/* Export public symbols */
.global ASMInterruptionInitialization, interruptEnable, interruptDisable

#define AM335X_VECTOR_BASE_ADDR 0x40300000 // L3 OCMC memory address
// stack = 8kiB for each mode
#define irqStackTop (AM335X_VECTOR_BASE_ADDR+0x2000)
#define svcStackTop (AM335X_VECTOR_BASE_ADDR+0x6000)
#define abtStackTop (AM335X_VECTOR_BASE_ADDR+0x8000)
#define undStackTop (AM335X_VECTOR_BASE_ADDR+0x10000)


/* Constants declaration */


/* Initialized variables declation */
.data                                                   // Initialized variables
.align  8                                               // declaration

/*
* This macro prepare the system before the interruption handler
*/
.macro myMacro offset, vector
  sub lr,lr,#\offset
  stmfd sp!,{r0-r12,lr}
  ldr r0, =\vector
  ldr r1,=AM335X_VECTOR_BASE_ADDR
  ldr r1,[r1]
  add r1,#\offset
  ldr r2, interruptHandler
  blx r2
  ldmfd sp!,{r0-r12,pc}^
.endm

interruptVectorStart:                                   // interrupt vector
  b resetHandler
  b undefined_handler
  b software_handler
  b prefetch_handler
  b data_handler
  b reserved_handler
  b irq_handler

resetHandler:        b resetHandler
undefined_handler:     myMacro 0,0
software_handler:      myMacro 0,1
prefetch_handler:      Mymacro 4,2
data_handler:        Mymacro 4,3
reserved_handler:      b reserved_handler
irq_handler:       myMacro 4,4

interruptHandler: .long 0
interruptVectorEnd:

/* Uninitialized variables declation */
.bss
.align  8

/* Implementation of assembler functions and methods */
.text                                                   // Program start



interruptEnable:
  mrs r0, cpsr
  bic r0, #0x80                                         // bic = and not
  msr cpsr, r0
  bx lr

interruptDisable:
  mrs r0, cpsr
  orr r0, r0, #0x80
  msr cpsr, r0
  bx lr

ASMInterruptionInitialization:
  push {lr}
  msr cpsr_c, #0xd2                                     // Switch to IRQ mode,
    interrupt desactivated
  ldr sp,=irqStackTop                                   // Define sp for IRQ
  msr cpsr_c, #0xd7                                     // Switch to Abort mode
```

```
    ldr sp,=abtStackTop                                 // Define sp for Abort
    msr cpsr_c,#0xdb                                    // Switch to undef mode
    ldr sp,=undStackTop                                 // Define sp for undef
    msr cpsr_c,#0xd3                                    // Switch to supervisor mode
    ldr r1,=AM335X_VECTOR_BASE_ADDR
    mcr p15,#0,r1,c12,c0,#0                             // Define vector table address
    ldr r1, =interruptHandler
    str r0, [r1]                                        // interruptHandler =
      init1_handler
    ldr r0, =AM335X_VECTOR_BASE_ADDR                    // Load parameters for memcpy
    ldr r1, =interruptVectorStart
    ldr r2, =(interruptVectorEnd-interruptVectorStart)
    bl  memcpy                                          // Save vectors table
    pop {pc}
```

Listing 5 – interrupt_asm.S

Remarque : Les en-têtes des codes n'ont pas été affichés ci-dessus par soucis de lisibilité. Ils sont néanmoins bien présent dans les fichiers.



```
Low Level Interrupt Handling on ARM Cortex-A8
---------------------------------------------
Initialisation...
Initialisation done
---------------------------------------------

Test data abort with a miss aligned access
ARM Exception with vector 3 and param data abort

Test supervisor call instruction / software interrupt
ARM Exception with vector 1 and param software interrupt

Test a invalid instruction
ARM Exception with vector 0 and param undefined instruction

Test a prefetch abort. This method will never return...
ARM Exception with vector 2 and param prefetch abort
```

FIGURE 1 – Résultat du programme

Fribourg, le 13 mars 2017

Samuel Riedo   Pascal Roulin