```
#include "printf.h"
#include "register.h"
#include "interrupts.h"
#include "reboot.h"
#include "armtimer.h"
#include "gpio.h"
#include "uart.h"
#include "pi.h"

// should not get called.
void impossible_handler(unsigned pc) {
        printf("impossible exception at pc=0x%x\n", pc);
        reboot();
}

volatile int counter = 0;

/*
 * we have only enabled timer interrupts
 */
void interrupt_handler(unsigned pc) {
#if 0
        printf("pc = %x\n", pc);
        printf("cpsr=%b\n", GETCPSR());
        printf("spsr=%b\n", GETSPSR());
#endif
    armtimer_clear_interrupt();
    counter++;
}

void main(void) {
  int lit = 0;

  gpio_init();
  gpio_set_output(ACT);
  printf_init();

  armtimer_init(2000000); // 1s
  armtimer_set_prescalar(125);
  armtimer_enable();

  armtimer_enable_interrupt();
  interrupts_enable_basic(INTERRUPTS_BASIC_ARM_TIMER_IRQ);
  system_enable_interrupts();

  printf("cpsr=0x%x\n", GETCPSR());
  int last = 0;
  while(1) {
    if(last != counter) {
      last = counter;
      gpio_write( ACT, lit );
      lit = !lit;
      printf("received %d interrupts\n", last);
    }
  }
  reboot();
}
```

```
extern int __bss_start__;
extern int __bss_end__;
extern int _vectors;
extern int _vectors_end;

extern void main();

#define RPI_VECTOR_START 0x0

void _cstart() {
  /* Zero out BSS */
  int* bss = &__bss_start__;
  int* bss_end = &__bss_end__;
  while( bss < bss_end )
    *bss++ = 0;

  /* Copy in interrupt vector vector and FIQ handler */
  /* _vector and _vector_end are symbols defined in the interrupt
     assembly file, at the beginning and end of the vector and
     its embedded constants.*/
  int* vectorsdst = (int*)RPI_VECTOR_START;
  int* vectors = &_vectors;
  int* vectors_end = &_vectors_end;
  while (vectors < vectors_end)
    *vectorsdst++ = *vectors++;

  main();
}
```

```
.globl _start
_start:
    mov sp, #0x800000  @ Set SVC stack pointer
    bl _cstart
hang: b reboot

/*
 * Interrupt vectors.
 */
.globl _vectors
.globl _vectors_end

_vectors:
  ldr pc, _reset_asm
  ldr pc, _undefined_instruction_asm
  ldr pc, _software_interrupt_asm
  ldr pc, _prefetch_abort_asm
  ldr pc, _data_abort_asm
  ldr pc, _reset_asm
  ldr pc, _interrupt_asm
fast_interrupt_asm:
  ldr pc, _fast_asm

_reset_asm:                     .word impossible_asm
_undefined_instruction_asm:     .word impossible_asm
_software_interrupt_asm:        .word impossible_asm
_prefetch_abort_asm:            .word impossible_asm
_data_abort_asm:                .word impossible_asm
_interrupt_asm:                 .word interrupt_asm
_fast_asm:                      .word impossible_asm

_vectors_end:

interrupt_asm:
        mov sp, #0x8000 @ Set interrupt stack pointer
        sub lr, lr, #4

        push {r0-r12,lr}

        mov  r0, lr
        bl  interrupt_handler

        pop {r0-r12, lr}

        // jump back
        //subs    pc, lr, #0
        movs    pc, lr

impossible_asm:
  mov sp, #0x7000  @ Set interrupt stack pointer
  sub   lr, lr, #4
  bl    impossible_handler    @ C function
  b     reboot
```