SCALETTA LEZIONE SERT 15.01.2021 (E13)

*** Supporto ai GPIO ***

1 Uso dei GPIO (general-purpose I/O pin)
  1.1 Consultazione del manuale della scheda Beaglebone Black (BBB SRM)
      1.1.1 In Table 12 e 13 della sezione 7 vi e' la descrizione dei
            connettori di espansione P8 e P9
      1.1.2 Il ruolo di ciascun pin dei due connettori e' programmato via
            software impostando il 'mode' per ciascun pin
  1.2 Consultazione del capitolo 25 del manuale del chip AM335
      1.2.1 Disponibili 4 moduli (circuiti) di GPIO
      1.2.2 Ciascun modulo offre 32 linee di I/O (totale 128 linee)
            1.2.2.1 La BBB esporta su P8 e P9 solo 65 linee
            1.2.2.2 Quasi tutte le linee sono multiplexate con altri
                    dispositivi
      1.2.3 Ciascuna linea puo' essere utilizzata per
            1.2.3.1 Ingresso di segnali (capture)
            1.2.3.2 Uscita di segnali (drive)
            1.2.3.3 Interfaccia verso un tasto con debouncing
            1.2.3.4 Generazione di interruzione hardware
            1.2.3.5 Risveglio da stati a basso consumo (wake-up): GPIO0
      1.2.4 Per poter utilizzare i moduli GPIO e' necessario innanzi
            tutto programmare i segnali di clock
            1.2.4.1 CM_PER_GPIOx_CLKCTRL, vedi AM335 TRM 8.1.12.1.29
      1.2.5 Successivamente e' necessario impostare la funzionalita' di
            ciascun pin che si vuole utilizzare (ad esempio, IN o OUT)
            1.2.5.1 GPIOx_OE, vedi AM335 TRM 25.3.4.3
      1.2.6 Si deve disabilitare o abilitare la generazione di
            interruzioni per tutti i pin configurati come output
            1.2.6.1 GPIOx_IRQSTATUS_CLR_0, GPIOx_IRQSTATUS_CLR_1,
                    vedi AM335 TRM 25.3.4.2.1.1
      1.2.7 Infine e' necessario programmare il multiplexer per esportare
            i segnali dei pin su P8 e P9
            1.2.7.1 CM_CONF_GPMC_xxx, vedi BBB SRM 7.0 e AM335 TRM
                    9.3.1.50

2 Inizializzazione dei pin 3, 4, 5 e 6 del connettore P8 come output
  2.1 Corrispondono ai pin 6, 7, 2 e 3 del modulo GPIO1
  2.2 In SERT abbiamo gia' inizializzato il modulo perche' GPIO1
      controlla anche i 4 LED della BBB
  2.3 Aggiungiamo a bbb_cm.h definizioni per programmare il multiplexer

```
|#define CONTROL_MODULE_BASE             0x44e10000           |
|iomemdef(CM_CONF_GPMC_AD2, CONTROL_MODULE_BASE + 0x808);     |
|iomemdef(CM_CONF_GPMC_AD3, CONTROL_MODULE_BASE + 0x80c);     |
|iomemdef(CM_CONF_GPMC_AD6, CONTROL_MODULE_BASE + 0x818);     |
|iomemdef(CM_CONF_GPMC_AD7, CONTROL_MODULE_BASE + 0x81c);     |
```

  2.4 Verificare che in bbb_gpio.h siano gia' presenti le definizioni per
      programmare il registro di stato degli IRQ

```
|#define GPIO1_BASE           0x4804c000                      |
|iomemdef(GPIO1_OE,           GPIO1_BASE + 0x134);            |
|iomemdef(GPIO1_DATAOUT,      GPIO1_BASE + 0x13c);            |
|iomemdef(GPIO1_CLEARDATAOUT, GPIO1_BASE + 0x190);            |
|iomemdef(GPIO1_SETDATAOUT,   GPIO1_BASE + 0x194);            |
|iomemdef(GPIO1_IRQSTATUS_CLR_0, GPIO1_BASE + 0x3c);          | !!!
|iomemdef(GPIO1_IRQSTATUS_CLR_1, GPIO1_BASE + 0x40);          | !!!
```

  2.5 Aggiungiamo a init_gpio1() in init.c le istruzioni opportune

```
|static void init_gpio1(void)                                 |
|{                                                            |
| u32 mask = (1 << 21) | (1 << 22) | (1 << 23) | (1 << 24);   |
| mask |= (1 << 6) | (1 << 7) | (1 << 2) | (1 << 3);          | <<<
| iomem(CM_PER_GPIO1_CLKCTRL) = 0x40002;                      |
| iomem_low(GPIO1_OE, mask);                                  |
| iomem_high(GPIO1_IRQSTATUS_CLR_0, mask);                    |
| iomem_high(GPIO1_IRQSTATUS_CLR_1, mask);                    |
| iomem(CM_CONF_GPMC_AD2) = 0x0f;                             | <<<
| iomem(CM_CONF_GPMC_AD3) = 0x0f;                             | <<<
| iomem(CM_CONF_GPMC_AD6) = 0x0f;                             | <<<
| iomem(CM_CONF_GPMC_AD7) = 0x0f;                             | <<<
|}                                                           | <<<
```

      2.5.1 Abbiamo aggiunto a mask i nuovi pin da pilotare come uscite
      2.5.2 Le quattro nuove linee sono configurate come 'faster slew
            rate', output only, pulldown selected ma disabilito, mode 7

3 Verifica del funzionamento delle linee di uscita

```
  3.1 Aggiungere in init_gpio1() l'impostazione di tutte le uscite a 0
      +-------------------------------------------------------------+
      |gpio1_off(2);                                                |
      |gpio1_off(3);                                                |
      |gpio1_off(6);                                                |
      |gpio1_off(7);                                                |
      +-------------------------------------------------------------+
  3.2 Modificare heartbeat() in main.c:
      +-------------------------------------------------------------+
      |static void heartbeat(void *arg __attribute__ ((unused)))    |
      |{                                                            |
      |  [...]                                                      |
      |    gpio1_toggle_mask((1<<2)|(1<<6));                       | <<<
      |}                                                            |
      +-------------------------------------------------------------+
  3.3 Aggiungere gpio1_on(2) in main() dopo banner()
  3.4 Uso di un oscilloscopio per analizzare i segnali digitali
      3.4.1 Collegare le due sonde ai pin, rispettivamente, 3-4 e 5-6
            del connettore P8
      3.4.2 Visualizzare le due forme d'onda quadra in opposizione di
            fase

4 Analisi della frequenza del segnale del tick periodico
  4.1 Rimuovere da heartbeat() l'istruzione gpio1_toggle_mask()
  4.2 Rimuovere da main() l'istruzione gpio1_on(2)
  4.3 Modificare isr_tick():
      +-------------------------------------------------------------+
      |[...]                                                        |
      |gpio1_toggle_mask(1<<3);                                     | <<<
      |[...]                                                        |
      +-------------------------------------------------------------+
  4.4 Visualizzare la forma d'onda risultante
      4.4.1 Misurare con il cursore la frequenza dell'onda quadra
      4.4.2 Risulta essere pari a 0.640 Hz, ci aspettavamo circa 0.5 Hz!
            (ogni tick corrisponde ad una transizione, quindi due tick
            per periodo)
      4.4.3 controllando sulla documentazione del chip ARM 335 20.1.2.2
            "The DMTimer 0 functional clock is fixed to use the internal
            32 kHz RC Clock (CLK_RC32K)." -- affermazione non sembra vera
      4.4.4 Cercando con maggior attenzione nel manuale scopriamo che
            l'oscillatore in questione e' realizzato con una rete
            Resistenza-Condensatore, circuito poco preciso ed altamente
            dipendente dalla temperatura
            5.3.4.1 In pratica possiamo aspettarci che la frequenza reale
                    misurata vari tra 16 kHz e 60 kHz!
      4.4.5 Una possibilita' e' quella di "aggiustare" la frequenza del
            timer 0
            4.4.5.1 Ma questo ci costringerebbe a ricalibrare ogni nuova
                    scheda, ed anche a ricalibrare la stessa scheda se le
                    condizioni operative cambiano

*** Cambio del timer per il tick di sistema ***

5 Cambio di timer per il tick di sistema
  5.1 La BBB dispone di un piu' affidabile oscillatore interno a 32768 Hz
      basato su un quarzo
  5.2 Il timer "DMTimer_1ms" puo' utilizzare questo oscillatore
  5.3 In piu' dispone di un meccanismo di correzione periodico della
      lunghezza del "tick" per tenere conto degli errori di troncamento
      introdotti nella programmazione del registro contatore
  5.4 Dettagli in AM335x TRM 20.2.3.1.1 e 20.2.4
  5.5 Cambio nome di bbb_timer.h in bbb_dmtimer0.h
      5.5.1 Rimuovere definizione macro HZ e TICK_TLDR da bbb_dmtimer0.h
      5.5.2 Rinominare le macro XXX_IT_FLAG in DMT0_XXX_IT_FLAG
  5.6 Creare un file bbb_dmtimer1.h
      +-----------------------------------------------------+
      |#define Timer1_Freq 32768    /* Hz */                |
      |#define Timer1_IRQ 67                                |
      |#define Timer1_IRQ_Bank (Timer1_IRQ/32)              |
      |#define Timer1_IRQ_Bit  (Timer1_IRQ%32)              |
      |#define Timer1_IRQ_Mask (1u<<Timer1_IRQ_Bit)         |
      |#define DMTIMER1_BASE        0x44e31000              |
      |iomemdef(DMTIMER1_TIDR,      DMTIMER1_BASE + 0x00);  |
      |iomemdef(DMTIMER1_TIOCP_CFG, DMTIMER1_BASE + 0x10);  |
      |iomemdef(DMTIMER1_TISTAT,    DMTIMER1_BASE + 0x14);  |
      |iomemdef(DMTIMER1_TISR,      DMTIMER1_BASE + 0x18);  |
      |iomemdef(DMTIMER1_TIER,      DMTIMER1_BASE + 0x1c);  |
      |iomemdef(DMTIMER1_TWER,      DMTIMER1_BASE + 0x20);  |
      |iomemdef(DMTIMER1_TCLR,      DMTIMER1_BASE + 0x24);  |
      |iomemdef(DMTIMER1_TCRR,      DMTIMER1_BASE + 0x28);  |
      |iomemdef(DMTIMER1_TLDR,      DMTIMER1_BASE + 0x2c);  |
```

```
|iomemdef(DMTIMER1_TTGR,      DMTIMER1_BASE + 0x30);  |
|iomemdef(DMTIMER1_TWPS,      DMTIMER1_BASE + 0x34);  |
|iomemdef(DMTIMER1_TMAR,      DMTIMER1_BASE + 0x38);  |
|iomemdef(DMTIMER1_TCAR1,     DMTIMER1_BASE + 0x3c);  |
|iomemdef(DMTIMER1_TSICR,     DMTIMER1_BASE + 0x40);  |
|iomemdef(DMTIMER1_TCAR2,     DMTIMER1_BASE + 0x44);  |
|iomemdef(DMTIMER1_TPIR,      DMTIMER1_BASE + 0x48);  |
|iomemdef(DMTIMER1_TNIR,      DMTIMER1_BASE + 0x4c);  |
|iomemdef(DMTIMER1_TCVR,      DMTIMER1_BASE + 0x50);  |
|iomemdef(DMTIMER1_TOCR,      DMTIMER1_BASE + 0x54);  |
|iomemdef(DMTIMER1_TOWR,      DMTIMER1_BASE + 0x58);  |
|#define DMT1_TCAR_IT_FLAG    0x4                     |
|#define DMT1_OVF_IT_FLAG     0x2                     |
|#define DMT1_MAT_IT_FLAG     0x1                     |
+-----------------------------------------------------+
```

5.7 Creare un nuovo file bbb_timer.h:

```
+-----------------------------------------------------+
|#include "bbb_dmtimer0.h"                           |
|#include "bbb_dmtimer1.h"                           |
|#define HZ          1000 /* Tick frequency (Hz) */  |
|#define TICK_TLDR   (0xffffffffu-(Timer1_Freq/HZ)+1) |
|#define CONFIG_TICK_ADJUST 1                        |
|#define TICK_V0     (Timer1_Freq/HZ)                |
|#define TICK_V1     (Timer1_Freq*1000*(1000/HZ))    |
|#define TICK_TPIR   (((TICK_V0+1)*1000000ul)-TICK_V1)|
|#define TICK_TNIR   ((TICK_V0*1000000ul)-TICK_V1)   |
+-----------------------------------------------------+
```

6 Inizializzazione di DMTimer1 e calibrazione del loop delay:

6.1 Scrivere la funzione calibrate_udelay() in delay.c:

```
+-------------------------------------------------------------------+
|unsigned int loops_per_usec;                                       |
|#define LOOPS_PER_USEC_SHIFT 10                                   |
|#define CALIBRATE_LOOP_DELAY 10000000ul                           |
|void calibrate_udelay(void)                                        |
|{                                                                  |
|    unsigned long t, d = CALIBRATE_LOOP_DELAY;                     |
|    irq_disable();                                                 |
|    /* start clocking the DMTimer1 module */                      |
|    iomem(CM_WKUP_TIMER1_CLKCTRL) = 0x2;                           |
|    data_sync_barrier();                                           |
|    /* give enough time to hardware circuits */                   |
|    loop_delay(10000);                                             |
|    /* select high precision 32768 Hz oscillator */              |
|    iomem(CM_DPLL_CLKSEL_TIMER1MS_CLK) = 4;                        |
|    data_sync_barrier();                                           |
|    loop_delay(10000);                                             |
|    /* soft reset the module */                                   |
|    iomem(DMTIMER1_TIOCP_CFG) = 0x2;                               |
|    /* wait for reset completed */                                |
|    while (iomem(DMTIMER1_TISTAT) & 0x1) == 0)                     |
|        data_sync_barrier();                                       |
|    /* disable interrupts */                                      |
|    iomem(DMTIMER1_TIER) = 0;                                      |
|    data_sync_barrier();                                           |
|    /* start the TIMER counter */                                 |
|    iomem(DMTIMER1_TCLR) = 0x1;                                    |
|    data_sync_barrier();                                           |
|    while (d-- > 0)                                                |
|        data_sync_barrier();                                       |
|    /* TCRR was just reset to zero,                               |
|     * and it wraps around in approx 36 hours.                     |
|     * There should be no risk of overflow here */                |
|    t = iomem(DMTIMER1_TCRR);                                      |
|    data_sync_barrier();                                           |
|    irq_enable();                                                  |
|    loops_per_usec = (((CALIBRATE_LOOP_DELAY/1000000ul)            |
|                      *Timer1_Freq)<<LOOPS_PER_USEC_SHIFT)         |
|                      / (float) t;                                 |
|    printf("Calibration: loops_per_usec=%2f\n",                   |
|           loops_per_usec/(float)(1u<<LOOPS_PER_USEC_SHIFT));     |
|}                                                                  |
+-------------------------------------------------------------------+
```

6.2 Scrivere la funzione udelay():

```
+-------------------------------------------------------------------+
|void udelay(unsigned int usec)                                     |
|{                                                                  |
|    unsigned long loops =                                          |
|                (usec*loops_per_usec)>>LOOPS_PER_USEC_SHIFT;       |
|    loop_delay(loops);                                             |
|}                                                                  |
```

```
    +-------------------------------------------------------------+
  6.3 Aggiungere definizioni per registri di controllo in bbb_cm.h:
    +-----------------------------------------------------+
    |#define CM_WKUP                       0x44e00400      |
    |iomemdef(CM_WKUP_CLKSTCTRL,           CM_WKUP + 0x00);|
    |iomemdef(CM_WKUP_TIMER1_CLKCTRL,      CM_WKUP + 0xc4);|
    |#define CM_DPLL                       0x44e00500      |
    |iomemdef(CM_DPLL_CLKSEL_TIMER1MS_CLK, CM_DPLL + 0x28);|
    +-----------------------------------------------------+

7 Modificare la funzione init_ticks() in tick.c:
  +-------------------------------------------------------+
  |void init_ticks(void)                                  |
  |{                                                      |
  |    irq_disable();                                     |
  |    if (register_isr(Timer1_IRQ, isr_tick)) {          |
  |        irq_enable();                                  |
  |        panic0();                                      |
  |    }                                                  |
  |    iomem(DMTIMER1_TLDR) = TICK_TLDR;                  |
  |    iomem(DMTIMER1_TIER) = DMT1_OVF_IT_FLAG;           |
  |#if CONFIG_TICK_ADJUST                                 |
  |    iomem(DMTIMER1_TPIR) = TICK_TPIR;                  |
  |    iomem(DMTIMER1_TNIR) = TICK_TNIR;                  |
  |#else                                                  |
  |    iomem(DMTIMER1_TPIR) = 0;                          |
  |    iomem(DMTIMER1_TNIR) = 0;                          |
  |#endif                                                 |
  |    iomem(INTC_ILR_BASE + Timer1_IRQ) = 0x0;           |
  |    iomem_high(INTC_MIR_CLEAR_BASE + 8 *               |
  |                      Timer1_IRQ_Bank, Timer1_IRQ_Mask);|
  |    iomem(DMTIMER1_TCLR) = 0x3; /* Auto-reload, start */ |
  |    loop_delay(10000);                                 |
  |    iomem(DMTIMER1_TTGR) = 1;                          |
  |    irq_enable();                                      |
  |}                                                      |
  +-------------------------------------------------------+
  7.1 Modificare la funzione _init() in init.c:
    +--------------------+
    |[...]               |
    |init_taskset();     |
    |calibrate_udelay(); | <<<
    |[...]               |
    +--------------------+
  7.2 Aggiungere i prototipi di calibrate_udelay() e udelay() in comm.h

8 Modificare la funzione isr_ticks() in tick.c:
  +---------------------------------------------+
  |static void isr_tick(void)                   |
  |{                                            |
  |    iomem(DMTIMER1_TISR) = DMT1_OVF_IT_FLAG; | <<<
  |    gpio1_toggle_mask(1<<3);                 |
  |    [...]                                    |
  +---------------------------------------------+

9 Misurazione della frequenza di heartbeat()
  9.1 La frequenza misurata dell'onda quadra dovrebbe ora essere 0,5 Hz

10 Misurazione della frequenza del tick periodico
   10.1 Rimuovere da heartbeat() la istruzione per generare le due onde
   10.2 Aggiungere a isr_tick() la stessa istruzione
        +-----------------------------------------------------+
        |static void isr_tick(void)                           |
        |{                                                    |
        |   iomem(DMTIMER1_TISR) = DMT1_OVF_IT_FLAG;          |
        |   gpio1_toggle_mask((1<<2)|(1<<6));                 |
        |   [...]                                             |
        +-----------------------------------------------------+
   10.3 Frequenza misurata: oscilla periodicamente tra 496 Hz e 500 Hz
   10.4 Stiamo osservando l'effetto della compensazione della durata del
        tick periodico effettuata dal DMTimer1
   10.5 Se ricompiliamo disabilitando il meccanismo (CONFIG_TICK_ADJUST
        definito a 0 in bbb_timer.h) la frequenza osservata e' tra
        511.3 Hz e 512.3 Hz
        10.5.1 Il tick ha una lunghezza all'incirca di 980 usec

11 Analisi del jitter del tick periodico
   11.1 Impostare l'oscilloscopio come segue:
        11.1.1 Trigger 'normale', sul fronte di salita
        11.1.2 Spostare il trigger all'indietro fino a visualizzare il
               fronte di discesa al centro dello reticolo
```

11.1.3 Scala dei tempi, div = 500 ns o 250 ns
11.1.4 Persistenza del display all'infinito
11.2 Misurare tramite il cursore l'ampiezza dell'area in cui cade il fronte di discesa
11.3.1 La maggior parte dei tick ha un jitter di 580 nanosecondi
11.3.2 Accuratezza del tick: errore inferiore al 0.6 per mille
11.4 Possibili cause del jitter:
11.4.1 Cache dei dati (D-Cache)
11.4.2 Cache delle istruzioni (I-Cache)
11.4.3 Memory Management Unit (MMU)
11.4.4 Operazioni effettuate dalla CPU in hypervisor modes

12 Analisi della precisione della funzione udelay()
12.1 Poiche' e' basata su un busy loop della CPU, dobbiamo disabilitare le interruzioni per avere misurazioni accurate
12.2 Inseriamo in main() il seguente codice:

```
banner();
gpio1_on(2);
irq_disable();
for (;;) {
   udelay(100);
   gpio1_toggle_mask((1<<2)|(1<<6));
}
```

12.2 Effettivamente la durata del ritardo e' di 100 usec
12.3 Il jitter e' di circa 20 nsec (con IRQ disabilitati)

13 Analisi delle prestazioni dello scheduler
13.1 Rimuovere il codice di misurazione della udelay() da main()
13.2 Rimuovere da isr_tick() gpio1_toggle_mask()
13.3 Modificare main() per creare un task
13.2.1 Periodo 1 tick e scadenza relativa 1 tick
13.2.2 Job inverte una linea di uscita ad ogni esecuzione

```
static void drive_gpio(void *arg __attribute__ ((unused)))
{
    static int state = 0;
    if (state & 1)
        gpio1_on(6);
    else
        gpio1_off(6);
    ++state;
}
[...]
if (create_task(drive_gpio, NULL, 1, 1, 1, FPR, "drive_gpio")==-1){
    puts("ERROR: cannot create task drive_gpio\n");
    panic1();
}
```

13.3 Risultato: periodo ~1025 Hz (~0,976 msec), jitter 560 nanosecondi

*** Analisi del WCET ***

14 Analisi del WCET di una funzione
14.1 Modificare main() per rimuovere il task drive_gpio
14.2 Scrittura della funzione factor_ticks():

```
const unsigned long small_primes[] =
    { 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47,
      53, 59, 61, 67, 71, 73, 79, 83, 89, 97 };
void factor_ticks(void *arg __attribute__ ((unused)))
{
    unsigned long v = ticks;
    int i;
    printf("%u: ", v);
    i = 0;
    while (v > 1 && i < 25) {
        unsigned long w = v / small_primes[i];
        if (v == w * small_primes[i]) {
            printf("%u ", small_primes[i]);
            v = w;
            continue;
        }
        ++i;
    }
    if (v > 1)
        printf("[%u]", v);
    putnl();
}
```

```
  14.3 Creare un task periodico FPR con job factor_ticks(), periodo 17 e
       priorita' 1
  14.4 Non compila perche' le divisioni intere non sono implementate da
       ARM
      14.4.1 Modificare il Makefile in modo da includere la libreria di
             codice contenente le divisioni intere:
    +-------------------------------------------------------------------+
    |GCCLIB=$(shell $(CC) -print-libgcc-file-name)                      |
    |[...]                                                              |
    |$(TARGET).elf: $(AOBJS) $(COBJS) $(TARGET).lds                     |
    |  (LD) -nostdlib -T $(TARGET).lds -o $@ $(AOBJS) $(COBJS) $(GCCLIB)|
    +-------------------------------------------------------------------+
      14.4.2 Aggiungere una funzione nulla raise() a main.c
      +-----------------------------------------------------------------+
      |int raise(int n) { return n; }                                   |
      +-----------------------------------------------------------------+
             14.4.2.1 Le funzioni di libreria incluse possono invocare
                      raise() per inviare un segnale al processo in
                      esecuzione -- ma in SERT non esistono segnali
  14.5 Provare la funzione
  14.6 Aggiungere gpio1_on(6) all'inizio della funzione e gpio1_off(6)
       al termine
  14.7 Analizzare il segnale per determinare il tempo d'esecuzione
      14.7.1 WCET misurato 2 millisecondi dopo un minuto
      14.7.2 WCET aumenta dopo diversi minuti di osservazione
      14.7.3 In realta' la misura non e' un vero WCET perche' non
             effettuata 'isolando' il job da altri task ed interruzioni
/*
vim: tabstop=4 softtabstop=4 expandtab list colorcolumn=74 tw=73
*/
```