Ho troppi indirizzi hardcoded, per rendere il codice leggibile uso le MACRO, ci metto gli elementi hardcoded e me ne "dimentico".
#define gpio1_mask(v) do{ iomem(GPIO1_DATAOUT) = (v);} while(0), è un modo per "evitare problemi" in C.
- creo bbb_led.h per dimenticare componenti hardcoed (accendi/spegni) e maschere (modalità in cui si accendono i led).
- Header di bbb_led.h in bbb.h (dopo gpio, poichè ne dipende). Così rendo MAIN più snello.
- Testo il tutto e funziona. A cosa possono essermi utili questi led? SEGNALI DI PANICO (errori irrecuperabili)
void panic0(void)
{ panic (s); } dove 's' è un pattern particolare (luci accede da sx a dx, viceverse, etc), e il prototipo va in comm.h

In ARM ho 7 tipi di eccezione, spesso gestite diversamente da INTEL.

scaletta.txt Thu Nov 03 20:02:53 2022 1

SCALETTA LEZIONE SERT 03.11.2022 (E06)

1 Aggiungere a _init() l'inizializzazione della tabella dei vettori di interruzione
- 1.1 Riprendere slides lez-E03b per modalita' gestione delle eccezioni
- 1.2 Consultazione del manuale "ARM Cortex-A8 TRM" per determinare la posizione della tabella dei vettori interruzione (3.2.25, 3.2.68)
- 1.3 Scrittura della funzione get_vectors_address() in bbb_vectors.h:

```
static inline u32 *get_vectors_address(void)
{
  u32 v;
  __asm__ __volatile__("mrc p15, 0, %0, c1, c0, 0\n":"=r"(v)::);
  if (v & (1 << 13))
      return (u32 *) 0xffff0000;
  __asm__ __volatile__("mrc p15, 0, %0, c12, c0, 0\n":"=r"(v)::);
  return (u32 *) v;
}
```

istruzioni macchina presa da manuale.

metti in %0 il registro che andrà in 'v'

Se bit è così settato:

else metto valore base del registro

- 1.4 Scrittura della funzione init_vectors() in init.c — lo inizializzo

Problema: ho 32 bit = 4 byte per codificare il punto a cui saltare. In questi 4 byte non riesco ad inserire l'istruzione di salto + codice salto

```
static void init_vectors(void)
{
    extern void _reset(void);
    volatile u32 *vectors = get_vectors_address();
#define LDR_PC_PC 0xe59ff018   carico registro a partire da altro registro
    vectors[0] = LDR_PC_PC; /* Reset / Reserved */
    vectors[1] = LDR_PC_PC; /* Undefined instruction */
    vectors[2] = LDR_PC_PC; /* Software interrupt */
    vectors[3] = LDR_PC_PC; /* Prefetch abort */
    vectors[4] = LDR_PC_PC; /* Data abort */
    vectors[5] = LDR_PC_PC; /* Hypervisor trap */
    vectors[6] = LDR_PC_PC; /* Interrupt request (IRQ) */
    vectors[7] = LDR_PC_PC; /* Fast interrupt request (FIQ) */
    vectors[8] = (u32) _reset;  /* Reset / Reserved */
    vectors[9] = (u32) panic0;  /* Undefined instruction */
    vectors[10] = (u32) panic0; /* Software interrupt */
    vectors[11] = (u32) panic0; /* Prefetch abort */
    vectors[12] = (u32) panic1; /* Data abort */
    vectors[13] = (u32) panic0; /* Hypervisor trap */
    vectors[14] = (u32) panic0;   /* Interrupt request (IRQ) */
    vectors[15] = (u32) panic0; /* Fast interrupt request (FIQ)*/
#undef LDR_PC_PC
}
```

Carico nel Program Counter il PC stesso, ma si riferisce a 2 istruzioni macchina sotto, perchè c'è prefetch. Quando vado da 'f' a vector[0] = LDR_PC_PC in realtà sto a vectors[8] (da f a lui sono 2 istruzioni macchina)

Per ARM posso usare anche i vectors da 8 a 15.

- 1.5 Aggiungere invocazione di init_vectors() in _init()
- 1.6 Prova del funzionamento: aggiungere asm("swi 0") in main() interruzione software
- 1.7 Sostituire panic0() con panic2() in vector[10] (così sono sicuro di controllare esattamente quello che volevo controllare)

2 Consultare i manuali per informazioni su come si programma la porta seriale "UART"
- 2.1 Il manuale "AM335x TRM" dedica piu' di 100 pagine alla programmazione delle porte seriali. Tre possibilita':
  - 2.1.1 Programmazione tramite DMA: nel trasferimento dei dati non e' coinvolta la CPU Direct Memory Access, difficile

2.1.2 Programmazione tramite interruzioni: la CPU trasferisce i dati, le interruzioni segnalano la presenza di dati in ricezione o la disponibilita' di spazio nel buffer di trasmissione

2.1.3 Programmazione tramite CPU polling: la CPU trasferisce i dati e controlla attivamente lo stato dei buffer di ricezione/trasmissione Basata su interrogazione della porta seriale.

2.2 Scegliere la strada piu' semplice: programmare la porta seriale tramite CPU polling. Determinare posizione e funzionamento dei registri di I/O della porta seriale, in particolare il registro di dati e quello che informa sullo stato del buffer di trasmissione

2.2.1 Registro THR (Transmit Hold Register)
Scrittura: dato da trasmettere su porta seriale (bit 0-7 del registro)

2.2.2 Registro LSR (Line Status Register) mi dice QUANDO posso scrivere
Bit 5: TXFIFOE (Trasmit Hold Register Empty)

2.3 Scrivere bbb_uart.h aggiungendo le definizioni occorrenti

```
#define UART0_BASE      0x44e09000      da Manuale AM353x
iomemdef(UART0_THR,         UART0_BASE + 0);
iomemdef(UART0_LSR,         UART0_BASE + 0x14);
#define LSR_TXFIFOE         (1u<<5) offset shiftato di 5, per avere il bit che mi interessa.
```

2.3.1 Aggiungere include bbb_uart.h in beagleboneblack.h

2.4 Scrivere file uart0.c

2.4.1 Funzione putc() per scrivere un singolo carattere

- Senza polling funziona, ma escluderlo vuol dire non avere garanzie.
- In questo modo stampa:
  SERT
  SERT
  1

cioè non ritorna a colonna 0.

```
int putc(int ch)Tipo char formalmente è un intero a 4 byte, anche se qui è di 8 bit
{           Ho "volatile"; non ottimizzato  Bit in posizione che mi interessa: può essere 0 occupato, o 1 scrivibile.
  while (!(iomem(UART0_LSR) & LSR_TXFIFOE))
      /* do nothing */ ;
  iomem(UART0_THR) = ch; mi interessano solo gli ultimi 8 bit
  return 1;"1" perchè ho scritto un carattere
}
```

2.4.2 Aggiungere extern per putc() in comm.h:
extern void putc(int);

2.4.3 Provare la funzione putc() modificando main()

2.4.4 Funzione puts() per scrivere una stringa di caratteri

```
int puts(const char *st)
{
  int v = 0;                    contatore caratteri scritti
  while (*st) {                 arriva fino a terminatore stiringa
      v += putc(*st);
      if (*st++ == '\n')        Se "*st++ ", cioè il carattere successivo, NON è /n terminatore allora
          v += putc('\r');
  }                             Ora ritorna a colonna 0, uso "/r" perchè porta seriale usa vecchio protocollo,
  return v;                     Adesso stampa correttamente.
}
```

leggo e passo al carattere successivo

2.4.5 Aggiungere extern per puts() in comm.h

2.5 Rinominare main() in endless_led_blinking()

2.6 Scrivere nuova main() che stampa su seriale ed invoca endless_led_blinking()

3 Scrivere la funzione putnl() per inserire 'new line':

```
int putnl(void)
{
    putc('\n');
    putc('\r');
    return 2;
}
```

4 Scrivere la funzione puth() in uart0.c per stampare un valore senza segno in esadecimale

Più semplice rispetto al decimale

```
int puth(unsigned long v)
{
    int i, d, w = 0;
    u32 mask;

    mask = 0xf0000000;
    for (i = 0; mask != 0; i += 4, mask >>= 4) {
        d = (v & mask) >> (28 - i);
        w += putc(d + (d > 9 ? 'a' - 10 : '0'));
    }
    return w;
}
```

da sx a dx, "f" è 1° byte, gli altri a 0.

Shifto di 4 verso dx (da sx a dx)

Il primo shift è di 28, poi secondo 24… in funzione di "i"

Azzero tutti i bit tranne quelli associati a 'v'

caso true

caso false, sommo 0 (da 0 a 9)

da 11 a 15 sommo 'a' - 10, ad esempio se d = 10 -> d = 10 + a - 10 = a, cioè 10 in base 16

==========

```
/*
vim: tabstop=4 softtabstop=4 expandtab list colorcolumn=74 tw=74
*/
```