The template starts with:

Timer 3 for debouncing the three buttons ---->//DEBOUNCING

Timer 2 for controlling the loudspeaker ---->//LOUDSPEAKER

RIT to control joysticks ---->//JOYSTICK

Potentiometer that, depending on how you turn it, lights up the corresponding LED ---->//POTENTIOMETER

Remember to set the proper NVIC_SetPriority() in the “lib_x.c ” of each ‘x’ peripheral.

All the peripherals enabled except from the ADC converter (It is connected to the loudspeaker and makes noise)

TIMERS

Standard timers

```
#define LPC_TIM0          ((LPC_TIM_TypeDef    *) LPC_TIM0_BASE    )
#define LPC_TIM1          ((LPC_TIM_TypeDef    *) LPC_TIM1_BASE    )
#define LPC_TIM2          ((LPC_TIM_TypeDef    *) LPC_TIM2_BASE    )
#define LPC_TIM3          ((LPC_TIM_TypeDef    *) LPC_TIM3_BASE    )
```

```
typedef struct {
  __IO uint32_t IR;     //pg 504 – table 427  Interrupt Register      (flags di match/capture)
```

__IO uint32_t TCR; // **pg 505 – table 428** Timer Control Register (CEN, CRST -> enable/reset timer)

__IO uint32_t TC; // Timer Counter (contatore principale)

__IO uint32_t PR; // Prescale Register (divide PCLK)

__IO uint32_t PC; // Prescale Counter (contatore del prescaler)

__IO uint32_t MCR; // **pg 507 – table 430** Match Control Register //(MR0I/MR0R/MR0S ... MR3I/MR3R/MR3S)

__IO uint32_t MR0; // **questa è una delle soglie comparate con il TC pg 507 paragrafo 21.6.7 (no table)**// Match Register 0

__IO uint32_t MR1; // Match Register 1

__IO uint32_t MR2; // Match Register 2

__IO uint32_t MR3; // Match Register 3

__IO uint32_t CCR; // **pg 508 – table 431** Capture Control Register

__I uint32_t CR0; // Capture Register 0

__I uint32_t CR1; // Capture Register 1

uint32_t RESERVED0[2];

__IO uint32_t EMR; // **pg 509 – table 432** External Match Register (controlla i pin MATn.x)

uint32_t RESERVED1[12];

__IO uint32_t CTCR; // **pg 505 – table 429** Count Control Register (Timer/Counter mode, sorgente CAPn.x)

} LPC_TIM_TypeDef;

RIT

#define LPC_RIT ((LPC_RIT_TypeDef *) LPC_RIT_BASE)

typedef struct

{

__IO uint32_t RICOMPVAL; // RIT Compare Value: valore di confronto; quando il contatore (RICOUNTER) “matcha” questo valore si genera l’evento (interrupt/reset/stop dipende da RICTRL) **RIT pg 512 table 435**

__IO uint32_t RIMASK; // RIT Mask: maschera sul confronto; permette di rendere il match “meno preciso” (confronta solo alcuni bit) per ottenere periodi più lunghi/approssimati

__IO uint8_t RICTRL; // RIT Control (8 bit): abilita RIT e configura cosa succede al match (es. enable, reset del counter su match, interrupt on match, stop on debug, ecc.) **RIT pg 513 table 437**

uint8_t RESERVED0[3]; // Riservato: padding per riallineare RICOUNTER a un indirizzo word-aligned (offset corretto nel memory map)

__IO uint32_t RICOUNTER; // RIT Counter: contatore che incrementa (tipicamente a PCLK); viene confrontato con RICOMPVAL secondo la maschera RIMASK **RIT pg 513 table 438**

} LPC_RIT_TypeDef;

SYSTICK

Systick pg 794

Control and Status Register	32 bits	0xE000E010

Bit	Type	Description
16	R	Read as 1 if counter reaches 0 since last time this register is read; clear to 0 when read or when current counter value (in Current Value Register) is cleared
2	R/W	**1 = use processor free running clock** 0 = use external reference clock (STCLK)
1	R/W	**1 = generate interrupt when timer reaches 0** 0 = Do not generate interrupt
0	R/W	**1 = enable SYSTICK Timer** **0 = disable SYSTICK Timer**

Reload Value Register	24 bits	0xE000E014

SYSTICK Reload Value Register stores the value to reload when timer reaches 0.

Current Value Register	24 bits	0xE000E018

SYSTICK Current Value Register stores the current value of the timer. Writing any number clears its content.

GPIO

#define LPC_PINCON ((LPC_PINCON_TypeDef *) LPC_PINCON_BASE)

typedef struct {

__IO uint32_t PINSEL0; // – copre parte dei pin (tipicamente P0.0..P0.15)

__IO uint32_t PINSEL1; // **DAC pg 114,118 table 81**

__IO uint32_t PINSEL2; // – copre pin della porta successiva (es. P1.x)

__IO uint32_t PINSEL3; // **ADC,joystick,potenziometro pg 114,120 table 83**

__IO uint32_t PINSEL4; //**per led/bottoni pg 114, 120 table 84**

__IO uint32_t PINSEL5; //

__IO uint32_t PINSEL6; //

__IO uint32_t PINSEL7; //

__IO uint32_t PINSEL8; //

__IO uint32_t PINSEL9; //

__IO uint32_t PINSEL10; // – include anche funzioni speciali (dipende dal datasheet)

uint32_t RESERVED0[5]; // Riservato: gap nell'address map per allineare i registri PINMODE agli offset corretti

__IO uint32_t PINMODE0; // Modalità resistor interna (2 bit per pin): pull-up / repeater / none / pull-down (per i pin gestiti da PINSEL0)

__IO uint32_t PINMODE1; // Modalità resistor interna (2 bit per pin) (per i pin gestiti da PINSEL1)

__IO uint32_t PINMODE2; // Modalità resistor interna (2 bit per pin)

__IO uint32_t PINMODE3; // Modalità resistor interna (2 bit per pin)

__IO uint32_t PINMODE4; // Modalità resistor interna (2 bit per pin)

__IO uint32_t PINMODE5; // Modalità resistor interna (2 bit per pin)

```
  __IO uint32_t PINMODE6;     // Modalità resistor interna (2 bit per pin)
  __IO uint32_t PINMODE7;     // Modalità resistor interna (2 bit per pin)
  __IO uint32_t PINMODE8;     // Modalità resistor interna (2 bit per pin)
  __IO uint32_t PINMODE9;     // Modalità resistor interna (2 bit per pin)

  __IO uint32_t PINMODE_OD0;  // Open-drain porta P0 (1 bit per pin): 0=push-pull, 1=open-drain
  __IO uint32_t PINMODE_OD1;  // Open-drain porta P1 (1 bit per pin)
  __IO uint32_t PINMODE_OD2;  // Open-drain porta P2 (1 bit per pin)
  __IO uint32_t PINMODE_OD3;  // Open-drain porta P3 (1 bit per pin)
  __IO uint32_t PINMODE_OD4;  // Open-drain porta P4 (1 bit per pin)

  __IO uint32_t I2CPADCFG;    // Configurazione elettrica speciale dei pad I2C (SDA/SCL): drive/slew/filter (dipende dal datasheet)

} LPC_PINCON_TypeDef;
```

```
#define LPC_GPIO0           ((LPC_GPIO_TypeDef     *) LPC_GPIO0_BASE    )
#define LPC_GPIO1           ((LPC_GPIO_TypeDef     *) LPC_GPIO1_BASE    )
```
joystick
```
#define LPC_GPIO2           ((LPC_GPIO_TypeDef     *) LPC_GPIO2_BASE    )
```
led
```
#define LPC_GPIO3           ((LPC_GPIO_TypeDef     *) LPC_GPIO3_BASE    )
#define LPC_GPIO4           ((LPC_GPIO_TypeDef     *) LPC_GPIO4_BASE    )
```

```
typedef struct {

  union {
    __IO uint32_t FIODIR;      // led, bottoni, joystick, DAC (solo debug)  pg 132, 133 table 104-105
    struct {
      __IO uint16_t FIODIRL;   // Direzione pin 0..15 (low halfword)
      __IO uint16_t FIODIRH;   // Direzione pin 16..31 (high halfword)
    };
    struct {
//qui per ogni GPIO tu dividi il #pin di  x.#pin in 4 byte(perché sono 32
// possibili valori). Mentre hai solamente 5 possibili GPIO (corrispondono alla
//x della notazione di sopra) pg 134 – table 105
      __IO uint8_t  FIODIR0;   // Direzione pin 0..7   (byte 0)
      __IO uint8_t  FIODIR1;   // Direzione pin 8..15  (byte 1)
      __IO uint8_t  FIODIR2;   // Direzione pin 16..23 (byte 2)
      __IO uint8_t  FIODIR3;   // Direzione pin 24..31 (byte 3)
    };
  };

  uint32_t RESERVED0[3];       // Riservato: spazio per allineare i registri successivi all'offset corretto
```

```
  union {
    __IO uint32_t FIOMASK;     // Maschera (1 bit per pin): 1=bit mascherato (ignorato) nelle operazioni su FIOPIN
    struct {
      __IO uint16_t FIOMASKL;  // Maschera pin 0..15
      __IO uint16_t FIOMASKH;  // Maschera pin 16..31
    };
    struct {
      __IO uint8_t  FIOMASK0;  // Maschera pin 0..7
      __IO uint8_t  FIOMASK1;  // Maschera pin 8..15
      __IO uint8_t  FIOMASK2;  // Maschera pin 16..23
      __IO uint8_t  FIOMASK3;  // Maschera pin 24..31
    };
  };

  union {
    __IO uint32_t FIOPIN;      // Valore pin (1 bit per pin): lettura=stato pin; scrittura=imposta uscite (attenzione a non sovrascrivere altri bit)
```

//corrisponde alla parte #pin dei bit di x.#pin, per esempio 2.14 pg 138 – table 111. Qui leggi il valore di joystick, led, bottoni

```
    struct {
      __IO uint16_t FIOPINL;   // Stato/valore pin 0..15
      __IO uint16_t FIOPINH;   // Stato/valore pin 16..31
    };
    struct {
      __IO uint8_t  FIOPIN0;   // Stato/valore pin 0..7
```

```
    __IO uint8_t  FIOPIN1;   // Stato/valore pin 8..15
    __IO uint8_t  FIOPIN2;   // Stato/valore pin 16..23
    __IO uint8_t  FIOPIN3;   // Stato/valore pin 24..31
  };
};

union {
  __IO uint32_t FIOSET;      // Set uscite (write 1): scrivere 1 su un bit porta il
//pin a 1; scrivere 0 non ha effetto  led pg 132, 135
  struct {
    __IO uint16_t FIOSETL;   // Set pin 0..15
    __IO uint16_t FIOSETH;   // Set pin 16..31
  };
  struct {
    __IO uint8_t  FIOSET0;   // Set pin 0..7
    __IO uint8_t  FIOSET1;   // Set pin 8..15
    __IO uint8_t  FIOSET2;   // Set pin 16..23
    __IO uint8_t  FIOSET3;   // Set pin 24..31
  };
};

union {
  __O  uint32_t FIOCLR;      // Clear uscite (write 1): scrivere 1 su un bit porta il
//pin a 0; scrivere 0 non ha effetto (registro write-only) led pg 132, 136
  struct {
    __O  uint16_t FIOCLRL;   // Clear pin 0..15
```

```
    __O  uint16_t FIOCLRH;   // Clear pin 16..31
  };
  struct {
    __O  uint8_t  FIOCLR0;   // Clear pin 0..7
    __O  uint8_t  FIOCLR1;   // Clear pin 8..15
    __O  uint8_t  FIOCLR2;   // Clear pin 16..23
    __O  uint8_t  FIOCLR3;   // Clear pin 24..31
  };
 };

} LPC_GPIO_TypeDef;
```

ADC

```
#define LPC_ADC             ((LPC_ADC_TypeDef       *) LPC_ADC_BASE      )
```

```
typedef struct
{
  __IO uint32_t ADCR;       // A/D Control Register: configura canali, clock, modalità (burst), start trigger, power-up ADC ADC pg 588 table 532
  __IO uint32_t ADGDR;      // A/D Global Data Register: ultimo risultato “globale” + flags (DONE/OVERRUN) + canale che ha prodotto il dato ADC pg 589 table 533
       uint32_t RESERVED0;  // Riservato: non usare (padding per allineamento mappa registri)
```

__IO uint32_t ADINTEN; // A/D Interrupt Enable: abilita quali canali (e/o global) generano interrupt a fine conversione **ADC pg 589-590 table 534**

__I uint32_t ADDR0; // A/D Channel 0 Data Register: risultato + DONE/OVERRUN specifici del canale AD0.0

__I uint32_t ADDR1; // A/D Channel 1 Data Register: risultato + DONE/OVERRUN specifici del canale AD0.1

__I uint32_t ADDR2; // A/D Channel 2 Data Register: risultato + DONE/OVERRUN specifici del canale AD0.2

__I uint32_t ADDR3; // A/D Channel 3 Data Register: risultato + DONE/OVERRUN specifici del canale AD0.3

__I uint32_t ADDR4; // A/D Channel 4 Data Register: risultato + DONE/OVERRUN specifici del canale AD0.4

__I uint32_t ADDR5; // A/D Channel 5 Data Register: risultato + DONE/OVERRUN specifici del canale AD0.5 (es. VR1 su LandTiger)

__I uint32_t ADDR6; // A/D Channel 6 Data Register: risultato + DONE/OVERRUN specifici del canale AD0.6

__I uint32_t ADDR7; // A/D Channel 7 Data Register: risultato + DONE/OVERRUN specifici del canale AD0.7

__I uint32_t ADSTAT; // A/D Status Register: stato per canale (DONE/OVERRUN), utile per polling rapido senza leggere tutti gli ADDRx

__IO uint32_t ADTRM; // A/D Trim Register: calibrazione/trim ADC (raramente toccato in laboratorio; spesso lasciato default)

} LPC_ADC_TypeDef;

DAC

#define LPC_DAC ((LPC_DAC_TypeDef *) LPC_DAC_BASE)

typedef struct

{

__IO uint32_t DACR; // D/A Converter Register: scrivi qui il valore digitale da convertire in tensione (codice DAC + bit di configurazione) **DAC pg 594 table 540**

__IO uint32_t DACCTRL; // D/A Control Register: abilita/gestisce DMA e “double buffering”/timer per aggiornamenti automatici (se usati)

__IO uint16_t DACCNTVAL; // D/A Counter Value: valore di conteggio per il “timeout”/rate di update quando usi il meccanismo DMA/timeout

uint16_t RESERVED; // Riservato: padding/allineamento

} LPC_DAC_TypeDef;

SYSTEM CONTROL

#define LPC_SC ((LPC_SC_TypeDef *) LPC_SC_BASE)

typedef struct

{

__IO uint32_t FLASHCFG; // Configurazione Flash (wait states, ecc.)

```
    uint32_t RESERVED0[31];

  // PLL principale (PLL0)
  __IO uint32_t PLL0CON;      // Enable/disable PLL0, bypass
  __IO uint32_t PLL0CFG;      // Moltiplicatore/divisore PLL0
  __I  uint32_t PLL0STAT;     // Stato PLL0 (locked, M, N effettivi, ecc.)
  __O  uint32_t PLL0FEED;     // Registro "feed": sequenza di sblocco per cambiare PLL
    uint32_t RESERVED1[4];

  // PLL USB (PLL1)
  __IO uint32_t PLL1CON;
  __IO uint32_t PLL1CFG;
  __I  uint32_t PLL1STAT;
  __O  uint32_t PLL1FEED;
    uint32_t RESERVED2[4];

  __IO uint32_t PCON;         // Power Control: sleep, deep-sleep, power-down, flags sample.c pg 64 table 45
  __IO uint32_t PCONP;        // Power Control for Peripherals: abilita/disabilita clock ai periferici TIMER2-3  RIT ADC pg 65 table 46

    uint32_t RESERVED3[15];

  // Configurazione clock di sistema
  __IO uint32_t CCLKCFG;      // Divider per CCLK (CPU clock)
```

__IO uint32_t USBCLKCFG;   // Divider per clock USB

__IO uint32_t CLKSRCSEL;   // Sorgente di clock (IRC, main osc, RTC)

__IO uint32_t CANSLEEPCLR; // Per CAN: uscita da sleep

__IO uint32_t CANWAKEFLAGS; // Per CAN: sorgente del wake-up

uint32_t RESERVED4[10];

// External Interrupts EINT0..EINT3 (vedi che numero sono dal datasheet)

__IO uint32_t EXTINT;   // **bottoni pg 27 – table 10** Flag di interrupt esterni (da //azzerare in ISR)

uint32_t RESERVED5;

__IO uint32_t EXTMODE;   // **bottoni pg 28 – table 11** 0 = level sensitive, 1 = edge //sensitive per EINTx

__IO uint32_t EXTPOLAR;   // **bottoni pg 28 – table 12** Se edge: rising/falling; se level: //high/low active

uint32_t RESERVED6[12];

__IO uint32_t RSID;   // Reset Source ID: chi ha causato il reset (POR, WDT, ecc.)

uint32_t RESERVED7[7];

__IO uint32_t SCS;   // System Controls & Status (main osc enable/ready, fast GPIO)

__IO uint32_t IRCTRIM;   // Trim dell'oscillatore interno IRC

__IO uint32_t PCLKSEL0;   // Selezione divisori di clock per vari periferici (parte bassa) **TIMER0-1 pg 58 table 40 + pg 59 table 42**

```
  __IO uint32_t PCLKSEL1;     // Selezione divisori di clock per gli altri periferici
```

TIMER1-2 RIT pg 59 table 41 + pg 59 table 42

```
      uint32_t RESERVED8[4];

  __IO uint32_t USBIntSt;     // Stato interrupt USB globale
  __IO uint32_t DMAREQSEL;    // Selezione richieste DMA per alcune sorgenti
  __IO uint32_t CLKOUTCFG;    // Clock Output: quale clock portare fuori su pin dedicato
          } LPC_SC_TypeDef;



typedef struct
{
  __IM  uint32_t CPUID;       // Offset 0x000: CPUID Base Register (identifica il core: implementer, variant, part number, revision)

  __IOM uint32_t ICSR;        // Offset 0x004: Interrupt Control and State Register
                              // (stato/controllo eccezioni: pending di SysTick/PendSV/NMI, vettore attivo, request pend, ecc.)

  __IOM uint32_t VTOR;        // Offset 0x008: Vector Table Offset Register
                              // (indirizzo base della vector table; permette relocation in RAM/Flash)
```

__IOM uint32_t AIRCR; // Offset 0x00C: Application Interrupt and Reset Control Register

// (priority grouping PRIGROUP + chiave VECTKEY, richiesta reset SYSRESETREQ, ecc.)

__IOM uint32_t SCR; // Offset 0x010: System Control Register

// (sleep/deep sleep e comportamento in sleep: SLEEPONEXIT, SLEEPDEEP, SEVONPEND) **sample.c pg 784 table 662**

__IOM uint32_t CCR; // Offset 0x014: Configuration Control Register

// (opzioni core: allineamento stack, trap div-by-zero, unaligned access trap, ecc.)

__IOM uint8_t SHPR[12U]; // Offset 0x018: System Handlers Priority Registers

// (priorità delle eccezioni “di sistema” 4..15: MemManage, BusFault, UsageFault, SVCall, PendSV, SysTick, ...)

__IOM uint32_t SHCSR; // Offset 0x024: System Handler Control and State Register

// (enable e stato fault: MemManage/BusFault/UsageFault enable, pending/active, ecc.)

__IOM uint32_t CFSR; // Offset 0x028: Configurable Fault Status Register

// (flag dettagliati di MemManageFault + BusFault + UsageFault)

__IOM uint32_t HFSR; // Offset 0x02C: HardFault Status Register

// (cause “hard fault”, escalation, forced hardfault, vettore hardfault, ecc.)

__IOM uint32_t DFSR; // Offset 0x030: Debug Fault Status Register

// (eventi di debug: halt, breakpoint, watchpoint, external debug request, ecc.)

__IOM uint32_t MMFAR; // Offset 0x034: MemManage Fault Address Register

// (indirizzo che ha causato MemManage fault, se valido)

__IOM uint32_t BFAR; // Offset 0x038: BusFault Address Register

// (indirizzo che ha causato BusFault, se valido)

__IOM uint32_t AFSR; // Offset 0x03C: Auxiliary Fault Status Register

// (fault “ausiliari” implementazione-specifici; spesso non usato nei casi base)

__IM uint32_t ID_PFR[2U]; // Offset 0x040: Processor Feature Register

// (feature supportate dal core: Thumb, ecc. - read-only)

__IM uint32_t ID_DFR; // Offset 0x048: Debug Feature Register (capacità debug del core - read-only)

__IM uint32_t ID_AFR; // Offset 0x04C: Auxiliary Feature Register (feature ausiliarie - read-only)

```
  __IM  uint32_t ID_MMFR[4U];  // Offset 0x050: Memory Model Feature Registers (feature modello memoria/MPU - read-only)


  __IM  uint32_t ID_ISAR[5U];  // Offset 0x060: Instruction Set Attributes Registers (attributi ISA - read-only)


       uint32_t RESERVED0[5U];// Riservato: padding/alignment nel memory map SCB


  __IOM uint32_t CPACR;        // Offset 0x088: Coprocessor Access Control Register
                               // (abilita accesso a coprocessori; tipicamente FPU su core che la supportano)


       uint32_t RESERVED3[93U];// Riservato: spazio fino all'offset 0x200


  __OM  uint32_t STIR;         // Offset 0x200: Software Triggered Interrupt Register (write-only)
                               // (per generare via software un interrupt esterno specifico - utile per test)


} SCB_Type;
```

INTERRUPT FUNCTS

```
EINT0_IRQn    INT0
EINT1_IRQn    KEY1
EINT2_IRQn    KEY2
RIT_IRQn      RIT
TIMER0_IRQn   TIMER0
TIMER1_IRQn   TIMER1
TIMER2_IRQn   TIMER2
TIMER3_IRQn   TIMER3
ADC_IRQn      ADC
```

`void NVIC_EnableIRQ(IRQn_Type IRQn);`

Abilita un interrupt periferico (IRQn >= 0) nel NVIC (ISER).

`void NVIC_DisableIRQ(IRQn_Type IRQn);`

Disabilita un interrupt periferico (scrive su ICER).

`void NVIC_SetPriority(IRQn_Type IRQn, uint32_t priority);`

Imposta la priorità di un IRQ periferico (NVIC->IP) o di un'eccezione core (SCB->SHP).

`void NVIC_SetPriorityGrouping(uint32_t PriorityGroup);`

Imposta il "priority grouping" (split tra preemption priority e subpriority) in SCB->AIRCR (valori 0..7).

uint32_t NVIC_GetPriorityGrouping(void);

Legge il priority grouping corrente da SCB->AIRCR.

uint32_t NVIC_GetEnableIRQ(IRQn_Type IRQn);

Ritorna 1 se l'IRQ periferico è abilitato, 0 altrimenti (legge ISER).

uint32_t NVIC_GetPendingIRQ(IRQn_Type IRQn);

Ritorna 1 se l'IRQ è pending, 0 altrimenti (legge ISPR).

void NVIC_SetPendingIRQ(IRQn_Type IRQn);

Forza lo stato pending di un IRQ (scrive su ISPR).

void NVIC_ClearPendingIRQ(IRQn_Type IRQn);

Cancella lo stato pending di un IRQ (scrive su ICPR).

uint32_t NVIC_GetActive(IRQn_Type IRQn);

Ritorna 1 se l'IRQ è attualmente "active" (in esecuzione o preempted), 0 altrimenti (legge IABR).

uint32_t NVIC_GetPriority(IRQn_Type IRQn);

Legge la priorità di un IRQ periferico o eccezione core.

uint32_t NVIC_EncodePriority(uint32_t PriorityGroup, uint32_t PreemptPriority, uint32_t SubPriority);

Converte (group, preempt, sub) in un valore "priority" da passare a NVIC_SetPriority().

void NVIC_DecodePriority(uint32_t Priority, uint32_t PriorityGroup, uint32_t *pPreemptPriority, uint32_t *pSubPriority);

Fa l'inverso di Encode: da "Priority" ricava preempt priority e subpriority.

void NVIC_SetVector(IRQn_Type IRQn, uint32_t vector);

Scrive l’indirizzo dell’handler nella vector table puntata da SCB->VTOR (tipicamente dopo relocation in SRAM).

`uint32_t NVIC_GetVector(IRQn_Type IRQn);`

Legge l’indirizzo dell’handler dalla vector table (SCB->VTOR).

`__NO_RETURN void NVIC_SystemReset(void);`

Richiede un reset di sistema (scrive SYSRESETREQ in SCB->AIRCR) e non ritorna.