LETTURA E CONVERSIONE DATI DA SENSORE DI TEMPERATURA ATTRAVERSO ADC

Gli ADC del microcontrollore sono estremamente raffinati e configurabili. Si possono configurare la risoluzione (numero di bit), la frequenza di campionamento, la modalità di conversione (singola, continua e scan). L'ADC può funzionare in modalità polling, interrupt o DMA (l'ultimo anche pilotato da un timer). Una cosa interessante è che gli ADC sono multicanale. Un canale è una sorgente di input per l'ADC, e un ADC di solito ne ha più di uno. I canali possono essere letti singolarmente (ne leggo solo uno), oppure posso configurare l'ADC per leggere i canali in sequenza, secondo un ordine opportunamente stabilito (modalità multicanale). Sul libro di Carmine Noviello (sez. 12.2.1, pag. 398) vengono spiegati i vari modi di conversione:

- Singolo canale, singola conversione: viene letto un solo campione da un solo canale;
- Conversione scan singola: si configura un rank, ossia una sequenza di canali, e viene letto un singolo campione da ciascun canale nella sequenza; funziona solo in modalità DMA;
- Singolo canale, conversione continua: vengono letti in maniera continua, ad una certa frequenza, i valori su un certo canale; l'ADC converte questi valori senza intervento dalla CPU;
- Conversione scan continua: configurato un rank, viene letto un singolo campione da ciascun canale nella sequenza, dopo di che si riparte dall'inizio della sequenza, indefinitamente, il tutto ad una certa frequenza; funziona solo in modalità DMA.

Esistono poi delle modalità ancora più complesse, come l'injected mode, che permette di effettuare una conversione al verificarsi di un evento, possibilmente interrompendo una conversione regolare (ad es. scan continua), e i multi mode (dual, triple), in cui due o tre ADC lavorano in modalità master-slave.
Notare che in alcuni ADC (nel nostro microcontrollore l'ADC1) certi canali sono riservati alla lettura di valori interni al microcontrollore:

- Un canale dell'ADC1 è collegato al sensore di temperatura;
- Un altro canale, chiamato Vrefint, è collegata ad un generatore di tensione che genera un certo valore di tensione di riferimento, detto V_{refint}, di circa 1,2V (vedere più avanti per il suo uso per calibrare il sensore di temperatura);
- Un terzo canale, chiamato Vbat, è collegato alla alimentazione esterna e legge il suo valore di tensione V_{bat} (non ci interessa).

Il sensore di temperatura integrato nel microcontrollore:

- è estremamente impreciso per le misure assolute (lo consigliano per effettuare misure di differenze di temperatura), e
- misura la temperatura superficiale del microcontrollore, non la temperatura ambiente,

ma può andare bene per generare un po' di numeri.

Una volta acquisito il dato dal sensore (notare che è un dato adimensionale) occorre convertirlo in gradi Celsius. Il sensore è essenzialmente affine nella temperatura, e produce un voltaggio che viene campionato dall'ADC. Ci sono due modi per ricostruire la temperatura: il primo è quello suggerito dal reference manual (RM0410), ed è basato su pendenza e intercetta della retta. Il metodo è piuttosto impreciso dal momento che questi dati, che si possono trovare sul datasheet del microcontrollore (tabella 78), possono variare al variare del processo di produzione del chip. Il secondo metodo, citato su un'application note (AN3964) che però fa riferimento al microcontrollore STM32L1x, è basato su dei dati che sono invece derivati da una fase di calibrazione che viene

effettuata chip per chip in fabbrica, dati che poi vengono scritti nella memoria del microcontrollore stesso. Questo metodo è ovviamente più affidabile, e quindi lo usiamo. I dati di calibrazione sono le letture dell'ADC per due valori di temperatura riportati sul datasheet.
Un ulteriore aspetto di cui tener conto è la tensione di alimentazione. Se l'ADC ha una risoluzione ad n bit, allora una lettura della tensione al suo input produce un valore compreso tra 0 e 2^n - 1. La tensione di input deve essere compresa in un range che va da 0 V alla tensione di alimentazione dell'ADC: se l'input dell'ADC è 0V, allora l'ADC produce in output il valore 0, se l'input è uguale (o maggiore) della sua tensione di alimentazione, allora l'ADC produce in output il valore 2^n - 1. In mezzo produce tutti i valori intermedi. Il problema è che la tensione di alimentazione usata durante la fase di calibrazione (V_{CAL}) potrebbe essere diversa dalla tensione di alimentazione usata quando facciamo operare il sistema (V_{refext}). Occorre pertanto applicare un fattore di compensazione. Le formule da usare sono le seguenti:

$$TEMP' = TEMP \cdot \frac{CAL}{REFINT}$$

dove TEMP è il valore del dato letto dal sensore di temperatura, TEMP' è il valore del dato al quale è stato applicato il fattore di compensazione per la differenza delle tensioni di alimentazione dell'ADC, CAL è la lettura dell'ADC di un certo voltaggio di riferimento V_{refint} (che viene generato internamente dal microcontrollore in maniera molto riproducibile) effettuata in fase di calibrazione del sensore, quando l'ADC è alimentato con la tensione di calibrazione V_{CAL}, e REFINT è la lettura dell'ADC dello stesso voltaggio di riferimento V_{refint} effettuata nella fase di operazione dell'applicazione, quando l'ADC è alimentato con la tensione di operazione V_{refext}. CAL è memorizzata ad una certa locazione di memoria il cui indirizzo è nel datasheet (tabella 82, indicata come $V_{REFIN\ CAL}$), mentre REFINT va letta a runtime da ADC1 dal canale apposito, indicato con il nome Vrefint. Per farlo bisogna abilitare la scan conversion mode, ma la lettura in scan conversion mode va fatta obbligatoriamente in DMA. La seconda formula calcola la temperatura per interpolazione lineare:

$$temperature = 30 + \frac{80}{TEMP2 - TEMP1} \cdot (TEMP' - TEMP1)$$

dove temperature è la temperatura, 80 è la differenza di temperatura tra i due punti di temperatura usati nella fase di calibrazione del sensore, TEMP1 e TEMP2 sono i valori letti dal sensore nella fase di calibrazione rispettivamente con la temperatura bassa (30 °C) e la temperatura alta (110 °C), e sono anche essi memorizzati a certe locazioni di memoria i cui indirizzi sono nel datasheet (indicate come TSCAL1 e TSCAL2 in tabella 79).

Per il momento supponiamo che il fattore di compensazione sia circa uguale ad 1 (cosa che è vera quando la tensione di alimentazione usata a runtime è uguale alla tensione di alimentazione usata durante la calibrazione), e che quindi sia TEMP ≈ TEMP'. Possiamo quindi effettuare una lettura single channel e continuous del solo canale temperatura e lasciare perdere il canale Vrefint. Questo ci permette di effettuare la lettura in modalità interrupt.

Creiamo un nuovo progetto ed impostiamolo così nel configuratore:

- Analog > ADC1: abilitare temperature sensor channel.
- Parameter settings per ADC1: Continuous conversion mode: Enabled; End of Conversion selection: EOC flag at the end of all conversions.
- ADC_Settings > Clock prescaler > PCLK2 divided by 8, e sotto ADC_Regular_ConversionMode > Rank 1 > Sampling time a 480 cicli.
- NVIC settings per ADC1: abilitare NVIC.

L’ultima impostazione è perché la frequenza del campionamento dell’ADC1 deve rispettare il tempo minimo di acquisizione del dato di temperatura. Notare che l’ADC1 è collegato a PCLK2 e che per default questo è a 96 MHz, mentre il datasheet riporta (tabella 78) che il tempo minimo di campionamento per il sensore di temperatura è 10 µsec, e quindi la frequenza deve essere minore di 0.1 MHz. Eventualmente si può cercare di pilotare l’ADC attraverso un timer.

Altra operazione che effettuiamo è utilizzare il Serial Wire Viewer (SWV) per fare delle stampe diagnostiche sul debugger utilizzando la printf. A tale scopo occorre effettuare le seguenti operazioni:

- Aprire la debug configuration di Eclipse per il progetto corrente;
- Nel tab Debugger selezionare (dovrebbe essere già selezionato) “ST-LINK (ST-LINK GDB server)” come debug probe e SWD come interface (non JTAG);
- Più sotto, nel box “Serial Wire Viewer (SWV)” selezionare Enable; nel clock settings (molto importante!) inserire nella text box “Core clock” la frequenza di clock del core, come indicata nel clock tree view;
- Nella sezione Misc più sotto selezionare, se non già attiva, “Enable live expressions”.

Ora lanciare la debug configuration in modalità debug; quando il debugger si ferma aprire la view “SWV ITM Data Console”, e cliccare il pulsante dei settings  , quindi nella riga “Enable port” cliccare la checkbox della porta 0 e premere il pulsante Ok. Infine attivare il tracing premendo il pulsante  . Con il codice di seguito le printf stamperanno nella view “SWV ITM Data Console”.

main.c:

```
...
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include <stdio.h>
#include <string.h>

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

typedef enum {false = 0, true = 1} bool;

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
#define TS_CAL1_ADDR ((uint16_t *) 0x1ff0f44c)
#define TS_CAL2_ADDR ((uint16_t *) 0x1ff0f44e)
/* USER CODE END PD */
...
/* Private variables ---------------------------------------------------------*/
...
/* USER CODE BEGIN PV */
static uint16_t ts_cal1;
static uint16_t delta_ts_cal;
/* USER CODE END PV */
...

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
int _write(int file, char *ptr, int len) {
  for (int i = 0; i < len; ++i) {
    ITM_SendChar(*ptr++);
```

```
  }
  return len;
}

/* USER CODE END 0 */
int main(void)
{
  ...
  /* USER CODE BEGIN 2 */

  ts_cal1 = *TS_CAL1_ADDR;
  uint16_t ts_cal2 = *TS_CAL2_ADDR;
  delta_ts_cal = ts_cal2 - ts_cal1;
  HAL_ADC_Start_IT(&hadc1);

  /* USER CODE END 2 */
  ...
}

...

/* USER CODE BEGIN 4 */


void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc) {
  if (hadc->Instance == ADC1) {
    uint32_t value = HAL_ADC_GetValue(&hadc1);
    float temperature = ((float) ((((int) value) - ts_cal1) * 80)) /
delta_ts_cal + 30;
    printf("t = %f C\r\n", temperature);
  }
}
/* USER CODE END 4 */
```

Il compilatore si lamenterà che non riesce ad effettuare la printf dei valori float: occorre a tale scopo attivare un’opportuna opzione di compilazione (le librerie C usate a runtime hanno printf/scanf dei valori float disattivate). Da menu principale selezionare Project > Properties…, quindi nel dialogo sulla sinistra C/C++ Build > Settings. Nel pannello a destra selezionare MCU Settings, e attivare l’opzione “Use float with printf from newlib nano”.

Notare che l’acquisizione di ts_cal1, ts_cal2 e il calcolo di ts_cal2 - ts_cal1 sono effettuati nel main, dal momento che non cambiano da iterazione a iterazione e quindi è inutile ripeterli. Notare che è necessario convertire value in int dal momento che è unsigned, e che quindi la sottrazione di ts_cal1 (che potrebbe essere più grande di value) potrebbe generare un valore negativo. In generale, dal momento che value non assume valori maggiori di poche migliaia, il tipo di dato intero a 32 bit è abbastanza grande per contenere i risultati intermedi, anche se faccio prima la moltiplicazione e poi la divisione.