UNIVERSITÀ DEGLI STUDI DELL'INSUBRIA

DIPARTIMENTO DI SCIENZE TEORICHE E APPLICATE
CORSO DI STUDIO TRIENNALE IN INFORMATICA - F004

Sviluppo di un sistema embedded per il controllo della temperatura in una camera di collaudo

Relatore:
Prof. Carlo Dossi

Tesi di Laurea di:
Mattia Papaccioli - 747053

Tutor Aziendale:
Edoardo Scaglia

Azienda ospitante:
AMEL SRL

Anno Accademico:
2025/2026

Indice

1

Introduzione

1.1 Abstract

La camera di collaudo, situata presso AMEL s.r.l., è utilizzata per testare carichi resistivi. Questi producono calore, rendendo necessario il raffreddamento dell'ambiente, soprattutto nei mesi più caldi.

Il sistema embedded controlla la frequenza di una ventola di raffreddamento mediante un controllo PID:

1. La temperatura ambiente viene letta tramite un sensore.
2. Questa viene utilizzata come ingresso del sistema PID.
3. L'algoritmo PID calcola la tensione da inviare a un inverter, il quale determina la frequenza della ventola in base all'errore attuale (azione proporzionale) e a quello accumulato (azione integrativa).
4. Questo sistema di retroazione negativa viene applicato continuamente per mantenere costante la temperatura ambiente.

La comunicazione tra l'inverter e il sistema embedded avviene tramite il protocollo MODBUS RTU.

Il sistema embedded fornisce inoltre un'interfaccia per regolare la temperatura target dell'ambiente mediante un display touchscreen. La GUI è sviluppata utilizzando la libreria LVGL.

Il sistema embedded è collegato alla rete aziendale tramite Ethernet; attraverso un web server sarà possibile regolare la temperatura target anche da remoto.

La comunicazione tra il sistema embedded e il web server avviene mediante scrittura su file o IPC (Inter-Process Communication).

È ancora da valutare l'impiego di un database per registrare la temperatura nel tempo. In tal caso, il web server gestirà l'interazione con esso.

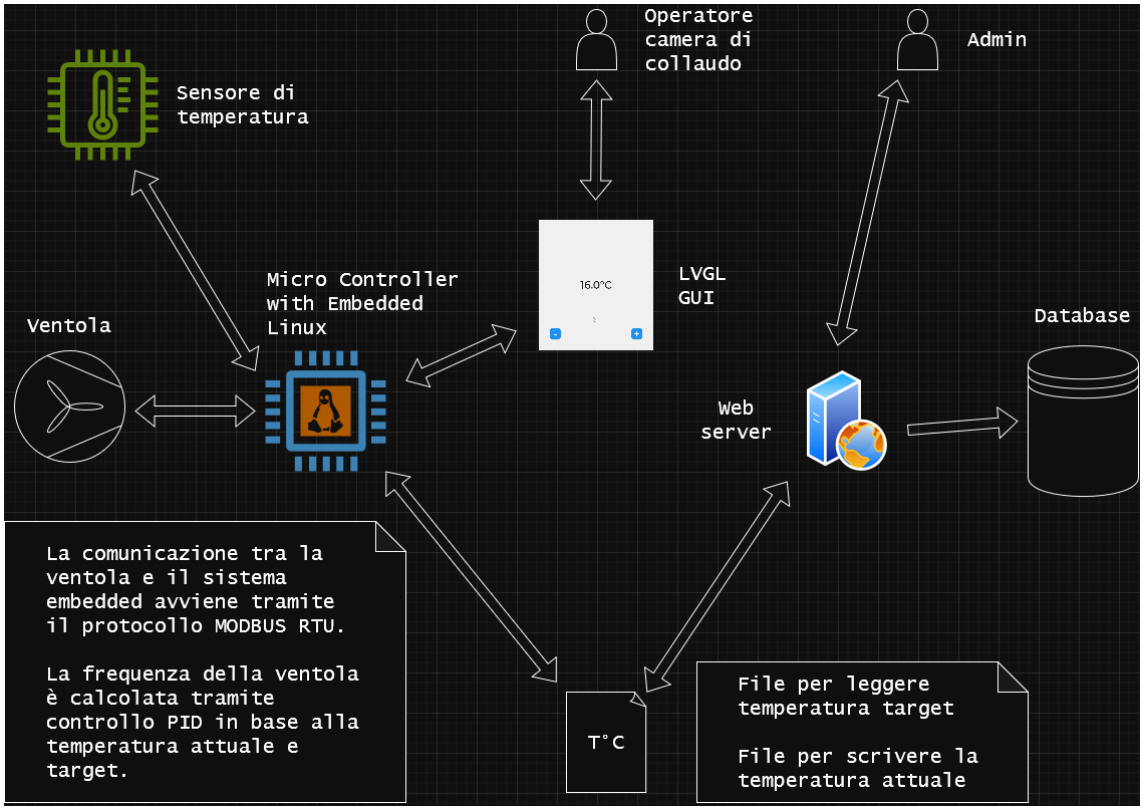

Figura 1 — Diagramma che illustea il sistema

1.2 Struttura del codice sorgente

Il codice sorgente del progetto è pubblicato su GitHub [1]. Il codice sorgente è composta da più moduli:

- `temp-control`: responsabile per la gui scritta in LVGL.
- `pid-control`: responsabile per il controllo pid, la rilevazione della temperatura e la comunicazione MODBUS rtu.
- `common-control`: contente funzioni helper comuni di aiuto e definizioni per la comunicazione tra interfaccia grafica e backend.

E stato intrapreso questo approccio per garantire modularita del codice, consentendo in futuro di poter rimpiazzare ciascun modulo con relativa facilita, rispetto ad un approccio monolitico.

2

Moduli

2.1 `common-control`

2.1.1 Comunicazione tra GUI e PID

Per fare in modo che i moduli `temp-control` e `pid-control` comunichino tra di loro e stato necessario sviluppare un sistema di segnali e scrittura e lettura su file.

Quando un operatore cambia la temperatura target dall'interfaccia sul display LCD essa viene scritta sul file `/opt/amel/target-temperature`.

Analogamente, il processo pid quando rileva una temperatura tramite i sensori DS18B20, scrive quest'ultima sul file `/opt/amel/current-temperature/sX`, con x che rappresenta il numero del sensore sul bus. Subito dopo aver scritto, viene mandato un segnale a `temp-control`, che a sua volta legge il file e aggiorna la temperatura del sensore spiegato successivamente.

2.1.2 `logging.h`

Per stampare a schermo in modo ordinato i messaggi dell'applicazione e stato implementata una semplice libreria in c che consente di loggare, anche con strighe formattate, ai livelli `TRACE`, `DEBUG`, `INFO`, `WARN`, `ERROR` e `FATAL`. Si puo decidere a che livello filtrare i messaggi e anche se scrivere su file nel syslog o sulla console.

Per evitare che piu log si sovrascrivino a vicenda viene utilizzato un meccanismo di mutex. E stata aggiunta un opzione per decidere la precisione del timer per intervalli di tempo sotto al secondo, utile per debuggare la regolarita del controllo pid.

2.1.3 `bash` templating with c preprocessor

Per evitare duplicazioni di costanti all'interno del modulo, e stato utilizzato il preprocessore del linguaggio c in modo creativo. Avendo definito le configurazioni in `include/config.h`, sono state utilizzate come input per dei template dal quale si ricavano gli script di inizializzazione e di avvio dell'intero progetto, rispettivamente `init.sh` e `run.sh`. Facendo compilare `gcc` con la flag `-E` possiamo sfruttare il preprocessore senza effettuare compilazione, assemblaggio e linking.

In questo modo e necessario cambiare le variabili solamente nell'header di configurazione per averle aggiornate anche negli script.

2.2 `temp-control`

Per controllare la temperatura della camera di collaudo, l'operatore imposta la temperatura target mediante un display touchscreen LCD «NOME DISPLAY». L'interfaccia grafica è sviluppata utilizzando la libreria LVGL [2] e si è utilizzato un template [3] contenente il porting su Linux fornito dagli sviluppatori della libreria.

Nell'interfaccia vengono mostrate temperatura target, temperatura attuale dei sensori collegati sul bus one-wire e due bottoni per aumentare e diminuire la temperatura target.

```
void set_target_temperature(float t)
{
    LOG_DEBUG("debug callback -> %.1f\n", target_temperature);
    target_temperature += t;
    lv_label_set_text_fmt(target_temperature_label,
target_temperature_format, target_temperature);
    write_float_to_file(TARGET_TEMPERATURE_FILE, target_temperature);
    kill(pid_control_pid, SIGUSR1);
}

static void increment_temperature(lv_event_t * e)
{
    if(lv_event_get_code(e) != LV_EVENT_CLICKED) {
        return;
    }
    set_target_temperature(1);
}

static void decrement_temperature(lv_event_t * e)
{
    if(lv_event_get_code(e) != LV_EVENT_CLICKED) {
        return;
    }
    set_target_temperature(-1);
}
```

Codice 1 — Funzioni di callback per i bottoni di incremento e decremento della temperatura target

Quando viene premuto il pulsante per aumentare o diminuire la temperatura target, viene invocata una funzione di callback che si occupa di aggiornare la `label` con la temperatura target sullo schermo e di scriverla nell'apposito file.

I backend utilizzati da LVGL per l'I/O sono libevdev e il framebuffer device. Sono stati scelti per la loro semplicità e il ridotto utilizzo di risorse.

Libevdev [4] è una libreria che gestisce gli eventi di input: riceve i tocchi dal touchscreen e li passa all'interfaccia grafica.

Il framebuffer device è semplicemente il file `/dev/fb0`, scritto dalla GUI, che contiene il colore di ciascun pixel dello schermo.

2.2.1 Compilazione della GUI

Per la compilazione dell'applicazione è necessaria una toolchain adatta all'architettura ARM. Nel nostro caso, ci affidiamo al compilatore e alle librerie fornite da Buildroot.

`cross_compile_setup.cmake`

```
set(CMAKE_SYSTEM_NAME Linux)
set(CMAKE_SYSTEM_PROCESSOR arm)

set(tools ~/buildroot/output/host/bin/arm-buildroot-linux-gnueabihf-)
set(CMAKE_C_COMPILER ${tools}gcc)
set(CMAKE_CXX_COMPILER ${tools}g++)

set(EVDEV_INCLUDE_DIRS ~/buildroot/output/staging/usr/include/libevdev/)
set(EVDEV_LIBRARIES ~/buildroot/output/staging/usr/lib/libevdev.so)

set(BUILD_SHARED_LIBS ON)
```

Codice 2 — cross_compile_setup.cmake

Il comando `cmake -DCMAKE_TOOLCHAIN_FILE=./cross_compile_setup.cmake -B build -S .` genera i Makefile necessari per la cross-compilazione, che vengono poi eseguiti con `make -C build -j`.

LVGL viene compilata come libreria condivisa, mentre l'applicazione come eseguibile.

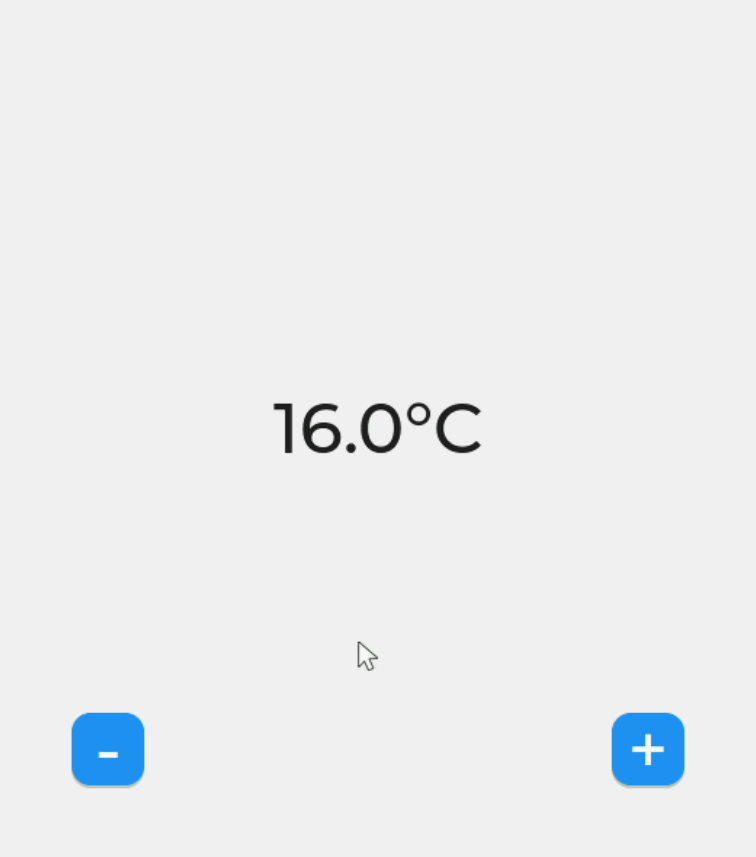

Figura 2 — Interfaccia grafica per il controllo della temperatura

2.2.2 Branches

Per organizzare efficientemente la repository sorgente, e stato realizzato un branching, creando una repository per lo sviluppo ed una per il dispositivo target. Esse sono uguali completamente tranne per il file `lv_conf.h`.

Per la branch di sviluppo, esso usa come backend `x11` e contiene dei sanity check, utili in sviluppo ma limitanti in termini di performance.

Per la branch del dispositivo target sono stati disabilitati i sanity checks e utilizzata come backend il device `/dev/fb0`.

Per proteggere il file `lv_conf.h` e stato aggiunto un file `.gitattributes` contenente `lv_conf.h merge=ours`.

Questo speciale file di git, comunica al version control system che durante il merge delle branch di mantenere il file come si trova nella branch da cui si sta effettuando il merge, consentendo di mantenere separate le due configurazioni senza preoccuparsi di sovrascriverle accidentalmente.

2.3 `pid-control`

2.3.1 Sensore di temperatura

I sensori di temperatura utilizzati sono due DS18B20 collegati in parallelo su un bus 1-Wire.

Il microcontrollore si comporta da master sul bus e richiede periodicamente la temperatura ai sensori.

Il binario `pid` legge periodicamente e calcola il valore di output del controller PID in base alla temperatura misurata e al setpoint desiderato.

Inizialmente esso conta il numero di sensori sul bus, alloca la memoria necessaria per immagazinare gli uuid dei sensori e poi legge effetivamente quest'ultimi in memoria.

E stato preferito questo approccio per evitare complicazioni con `realloc` rispetto a leggere direttamente in un ciclo unico sia il numero di sensori che gli id. Questa procedura viene effettuate solamente una volta all'avvio e non ha un impatto significativo sulla performance dell'eseguibile.

Un approccio senza salvare gli id dei sensori porterebbe una chiamata alla funzione `DS18X20_find_sensor` ripetutamente e sarebbe uno spreco di cicli di cpu quindi sacrifichiamo un po di memoria per questo.

```
typedef struct
{
    char id[16];
    int16_t temperature;
    uint8_t uint_id[OW_ROMCODE_SIZE];
} sensor;

void init_onewire_sensors()
{
    uint8_t diff = OW_SEARCH_FIRST;

    while (diff != OW_LAST_DEVICE)
    {
        sensors_count++;
        DS18X20_find_sensor(&diff, id);
    }

    write_char_to_file(NUMBER_OF_SENSORS_FILE, sensors_count);
    LOG_INFO("Found %d sensors on the 1-wire bus", sensors_count);

    sensors = malloc(sizeof(sensor) * sensors_count);

    diff = OW_SEARCH_FIRST;
    int i = 0;
    while (diff != OW_LAST_DEVICE)
    {
        DS18X20_find_sensor(&diff, id);
        sensor s;
        for (int i = 0; i < OW_ROMCODE_SIZE; i++)
        {
            s.uint_id[i] = id[i];
        }
        sprintf(s.id, "%02hx%02hx%02hx%02hx%02hx%02hx%02hx%02hx", id[0],
id[1],
               id[2], id[3], id[4], id[5], id[6], id[7]);
        s.temperature = 0;
        sprintf(s.file, CURRENT_TEMPERATURE_FILE "/s%d", i);
        LOG_INFO("sensor %d id %s", i, s.id);
        sensors[i] = s;
        i++;
    }
}
```

Codice 3 — `pid-main`

2.3.2 MODBUS RTU

Per comunicare con l'inverter che controlla la ventola di raffreddamento, è stato utilizzato il protocollo MODBUS RTU tramite l'apposita libreria `libmodbus`.

2.3.3 Controllo pid

Il controllo pid (Proporzionale, Integrale e Derivativo) e un sistema di retroazione negativa che permette di reagire ad un errore rispetto ad un valore target.

Esso viene utilizzato per mantenere la temperatura costante nella camera di collaudo. Prende in input la temperatura rilevate dai sensori e la temperatura desiderata all'interno della stanza e restituisce in output la tensione con la quale comunichiamo all'inverter la frequenza della ventola di raffreddamento.

Non e stato utilizzato il coefficiente derivativo perche

Monotonic clock

Per campionare la temperatura nella stanza ad una frequenza costante, fondamentale per un corretto calcolo PID, e stato utilizzato l'header `<sys/timerfd.h>` della `Standard C library`. Queste chiamate di sistema creano e operano su un timer che consegna segnali di scadenza del timer ad intervalli regolari tramite un file descriptor.

Scheduler priority

E stata assegnata la massima priorita di scheduler al programma tramite l'header `<sched.h>` per evitare interrupt durante la misurazione e per cercare di mantenere piu costante possibile la periodicita del campionamento.

2.3.4 Admin Control

2.3.5 Logging and Monitoring

3

Embedded Linux

3.1 Hardware

La scheda utilizzata per il sistema embedded è sviluppata da AMEL e comprende:

- una host-board Ganador (rev. 4) che integra:
  - una memoria SD utilizzata come disco fisso;
  - un'interfaccia Ethernet
  - un'interfaccia seriale
  - un display touchscreen
- un system-on-module Vulcano-A5 che integra:
  - una cpu Atmel ARM9 AT91SAM9X35 @ 400MHz
  - 128 MB DDR2 SDRAM
  - 256 MB NAND Flash
  - SODIMM200 interface
  - 10/100 Mbps Ethernet MAC Controller
  - 2x USB 2.0 Host, 1x USB 2.0 Host/Device
  - 3x USARTs
  - TFT LCD Controller with TTL / LVDS support

3.2 Costruzione del sistema

Per orchestrare il sistema è stato utilizzato Linux. È stato creato un kernel personalizzato con soli i moduli essenziali per il funzionamento, data la limitatezza delle risorse hardware. Lo strumento utilizzato per completare l'opera è Buildroot [5], che consiste essenzialmente in una serie di Makefile per installare e cross-compilare tutte le librerie e i pacchetti necessari alla costruzione e all'esecuzione del sistema. Inoltre, si occupa di creare il filesystem e di prepararlo in un'immagine pronta per essere scritta sulla scheda SD del sistema embedded.

Per aggiungere l'interfaccia grafica sviluppata con LVGL è stato necessario creare un nuovo pacchetto in Buildroot.

All'avvio del sistema, il bootloader del chip attiva AT91bootstrap, che a sua volta avvia Barebox, il quale carica il kernel in memoria.

3.3 Personalizzazione di buildroot

Per aggiungere un pacchetto a Buildroot è necessario inserire una nuova voce nella cartella `package`, comprendente un file `Config.in` e un file `.mk`.

Questi due file contengono le istruzioni che consentono a Buildroot di risolvere le dipendenze, scaricare e installare il pacchetto nel filesystem del dispositivo target.

3.3.1 Il pacchetto amel-temp-control

```
AMEL_TEMP_CONTROL_VERSION = 44c17c6f2c492f1f3c7d8a6767df390c8d13eb9c
AMEL_TEMP_CONTROL_SITE = git@git.amelchem.com:mpapaccioli/temp-control.git
AMEL_TEMP_CONTROL_SITE_METHOD = git

AMEL_TEMP_CONTROL_DEPENDENCIES = libevdev
AMEL_TEMP_CONTROL_GIT_SUBMODULES = YES

define AMEL_TEMP_CONTROL_BUILD_CMDS
  cmake -DCMAKE_TOOLCHAIN_FILE=$(@D)/user_cross_compile_setup.cmake \
        -B $(@D)/build -S $(@D)
  make -C $(@D)/build -j

endef

define AMEL_TEMP_CONTROL_INSTALL_TARGET_CMDS
  $(INSTALL) -d $(TARGET_DIR)/opt/amel-temp-control/
  cp $(@D)/build/bin/lvglsim $(TARGET_DIR)/opt/amel-temp-control/main
  cp -r $(@D)/build/lvgl/lib/* $(TARGET_DIR)/usr/lib

endef

$(eval $(generic-package))
```

Codice 4 — amel-temp-control.mk

Inizialmente, viene clonata la repository temp-control, contenente la GUI, al commit specificato, inizializzando i sottomoduli e verificando la presenza della dipendenza `libevdev`.

Successivamente, vengono cross-compilate la libreria LVGL e l'applicazione con interfaccia grafica utilizzando il compilatore ARM fornito da Buildroot.

Infine, i binari della libreria e l'eseguibile dell'applicazione vengono installati sulla macchina target.

3.3.2 Il pacchetto amel-pid

```
AMEL_PID_VERSION = v0.0.3
AMEL_PID_SITE = git@git.amelchem.com:mpapaccioli/pid.git
AMEL_PID_SITE_METHOD = git

AMEL_PID_DEPENDENCIES = libmodbus

define AMEL_PID_BUILD_CMDS
  make -C $(@D) CC=$(TARGET_CC)
endef

define AMEL_PID_INSTALL_TARGET_CMDS
  $(INSTALL) -d $(TARGET_DIR)/opt/amel-pid/

  cp $(@D)/pid $(TARGET_DIR)/opt/amel-pid/pid

endef

$(eval $(generic-package))
```

Codice 5 — amel-pid-control.mk

Analogamente, è stato creato il pacchetto `amel-pid-control` per cross-compilare e installare la libreria PID sviluppata in C++.

3.3.3 Modifica pacchetti networking

E stata creata una rete virtuale ed e stato aggiunto un ssh server per consentire il collegamento remoto al dispositivo embedded. E stato necessario modificare lo script in `/etc/init.d/S50network` per configurare l'interfaccia di rete virtuale `eth0` con un indirizzo IP statico all'avvio del sistema e permettere il login all'utente `root` tramite password modificando il file `/etc/ssh/sshd_config`. Dopo aver ripetuto questo processo per tutti i pacchetti desiderati, il filesystem e l'immagine del kernel vengono assemblati in un file `sdcard.img` pronto per essere scritto su una scheda SD e avviato sul dispositivo embedded. Inoltre, la rete virtuale e stata create in modo da condividere l accesso ad internet e dopo aver impostato come default gateway il pc creatore della rete bridge e possibile accedere alla rete globale dal sistema embedded. E stato utilizzato il dns server di google `8.8.8.8`. L'accesso ad internet e necessario per la configurazione di NTP, come descritto in seguito.

3.3.4 Installazione del modulo c210x per la porta seriale

Per consentire la comunicazione seriale tramite la porta RS-232 della host-board Ganador, è stato necessario includere il modulo kernel `c210x` per il controller USB-seriale. Il modulo e stato aggiunto tramite il menu di configurazione di Buildroot, selezionandolo in `Kernel modules -> USB Serial Converter support -> USB CP210x family of UART Bridge Controllers` dal comando `make linux-menuconfig`.

3.3.5 Il pacchetto NTP

Per garantire la corretta sincronizzazione dell-orologio di sistema e stato necessario installare il client `ntpd` per il Network Time Protocol tramite buildroot.

3.4 Custom patches a barebox e linux

A causa dei rebuild frequenti in fase di sviluppo, sono stati patchati i codici sorgenti di barebox e linux.

E stata effettuata una clonazione della repository del kernel linux di amel ed e stato abilitato di default il modulo cp210x, modificando il file `/drivers/usb/serial/Kconfig`.

Analogamente, e stato cambiato l ordine nel bootloader barebox, prioritarizzando il device `mmc2` invece che `nand`.

4

Conclusione

A Riferimenti

[1] M. Papaccioli, «Tesi». [Online]. Disponibile su: https://github.com/sbOogway/tesi

[2] L. Contributors, «LVGL». [Online]. Disponibile su: https://github.com/lvgl/lvgl

[3] L. Contributors, «Linux template for LVGL». [Online]. Disponibile su: https://github.com/lvgl/lv_port_linux

[4] [Online]. Disponibile su: https://www.freedesktop.org/wiki/Software/libevdev/

[5] B. contributors, [Online]. Disponibile su: https://buildroot.org/

B Glossario