



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

Tutor Aziendale:

Edoardo Scaglia

Tesi di Laurea di:

Mattia Papaccioli - 747053

Azienda ospitante:

AMEL SRL

Anno Accademico:

2025/2026

Indice

1	Introduzione	1
1.1	Abstract	1
1.2	Struttura del codice sorgente	2
2	Moduli	3
2.1	temp-control	3
2.1.1	Funzioni di callback nel ciclo principale della GUI LVGL	4
2.1.2	Albero dei widget nel ciclo principale	5
2.1.3	Compilazione della GUI	6
2.1.4	Branches	7
2.2	pid-control	8
2.2.1	Sensore di temperatura	8
2.2.2	MODBUS RTU	10
2.2.3	Controllo pid	10
2.3	common-control	10
2.3.1	Comunicazione tra GUI e PID	10
2.3.2	logging.h	10
2.3.3	Admin Control	10
2.3.4	Logging and Monitoring	10
3	Embedded Linux	11
3.1	Hardware	11
3.2	Costruzione del sistema	11
3.3	Personalizzazione di buildroot	12
3.3.1	Il pacchetto amel-temp-control	12
3.3.2	Il pacchetto amel-pid	13
3.3.3	Modifica pacchetti networking	13
3.3.4	Installazione del modulo c210x per la porta seriale	13
3.3.5	Il pacchetto NTP	14
3.4	Custom patches a barebox e linux	14
4	Conclusione	15
A	Riferimenti	16
B	Glossario	17

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.

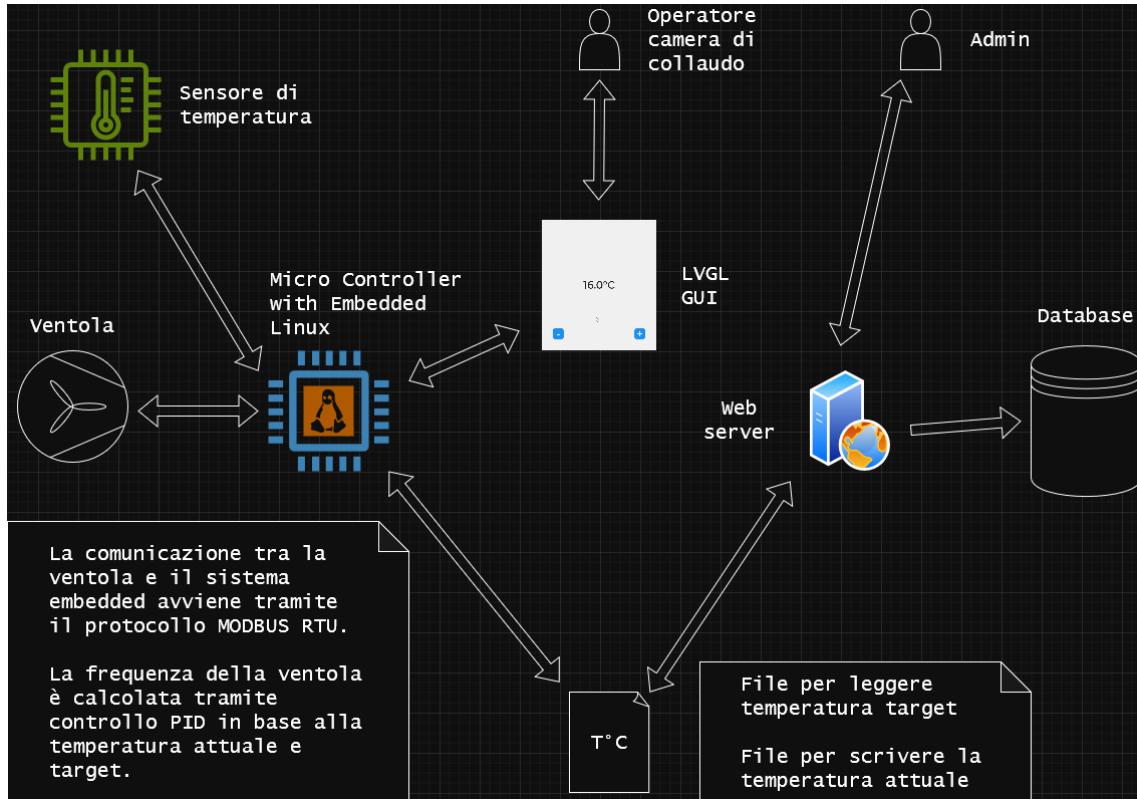


Figura 1 — Diagramma che illustra il sistema

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**: contiene funzioni helper comuni di aiuto e definizioni per la comunicazione tra interfaccia grafica e backend.

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

2

Moduli

2.1 temp-control

Per controllare la temperatura della camera di collaudo, l'operatore imposta la temperatura target mediante un display touchscreen «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.

2.1.1 Funzioni di callback nel ciclo principale della GUI LVGL

```
1 // TODO CHANGEME READ THIS FROM A FILE
2 static float target_temperature = read_temperature_file();
3 lv_obj_t * screen;
4 lv_obj_t * target_temperature_label;
5
6 const int padding_button = 50;
7 const int height_button = 50;
8 const int width_button = 50;
9
10 static void increment_temperature(lv_event_t * e)
11 {
12     if(lv_event_get_code(e) != LV_EVENT_CLICKED) {
13         return;
14     }
15     target_temperature++;
16     lv_label_set_text_fmt(
17         target_temperature_label, "%.1f°C", target_temperature
18     );
19 }
20
21 static void decrement_temperature(lv_event_t * e)
22 {
23     if(lv_event_get_code(e) != LV_EVENT_CLICKED) {
24         return;
25     }
26     target_temperature--;
27     lv_label_set_text_fmt(
28         target_temperature_label, "%.1f°C", target_temperature
29     );
30 }
```

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

2.1.2 Albero dei widget nel ciclo principale

```

1 screen = lv_scr_act();
2 lv_obj_t * increment_temperature_button = lv_btn_create(screen);
3 lv_obj_align(
4     increment_temperature_button, LV_ALIGN_BOTTOM_RIGHT,
5     -padding_button, -padding_button);
6 lv_obj_set_height(increment_temperature_button, height_button);
7 lv_obj_set_width(increment_temperature_button, width_button);
8 lv_obj_add_event_cb(
9     increment_temperature_button, increment_temperature,
10    LV_EVENT_ALL, NULL
11 );
12
13 lv_obj_t * increment_temperature_label = lv_label_create(
14     increment_temperature_button
15 );
16 lv_label_set_text(increment_temperature_label, "+");
17 lv_obj_set_style_text_font(
18     increment_temperature_label, &lv_font_montserrat_48, 0
19 );
20 lv_obj_center(increment_temperature_label);
21
22 lv_obj_t * decrement_temperature_button = lv_btn_create(screen);
23 lv_obj_align(
24     decrement_temperature_button, LV_ALIGN_BOTTOM_LEFT,
25     padding_button, -padding_button);
26 lv_obj_set_height(decrement_temperature_button, height_button);
27 lv_obj_set_width(decrement_temperature_button, width_button);
28 lv_obj_add_event_cb(
29     decrement_temperature_button, decrement_temperature,
30    LV_EVENT_ALL, NULL
31 );
32
33 lv_obj_t * decrement_temperature_label = lv_label_create(
34     decrement_temperature_button
35 );
36 lv_label_set_text(decrement_temperature_label, "-");
37 lv_obj_set_style_text_font(
38     decrement_temperature_label, &lv_font_montserrat_48, 0
39 );
40 lv_obj_center(decrement_temperature_label);
41
42 target_temperature_label = lv_label_create(screen);
43 lv_label_set_text_fmt(
44     target_temperature_label, "%.1f°C", target_temperature
45 );
46 lv_obj_set_style_text_font(
47     target_temperature_label, &lv_font_montserrat_48, 0
48 );
49 lv_obj_align(target_temperature_label, LV_ALIGN_CENTER, 0, 0);

```

Codice 2 – Creazione dei widget nell'interfaccia grafica

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.

2.1.3 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`

```
1 set(CMAKE_SYSTEM_NAME Linux)
2 set(CMAKE_SYSTEM_PROCESSOR arm)
3
4 set(tools ~/buildroot/output/host/bin/arm-buildroot-linux-gnueabihf-)
5 set(CMAKE_C_COMPILER ${tools}gcc)
6 set(CMAKE_CXX_COMPILER ${tools}g++)
7
8 set(EVDEV_INCLUDE_DIRS ~/buildroot/output/staging/usr/include/libevdev/)
9 set(EVDEV_LIBRARIES ~/buildroot/output/staging/usr/lib/libevdev.so)
10
11 set(BUILD_SHARED_LIBS ON)
```

Codice 3 – `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.

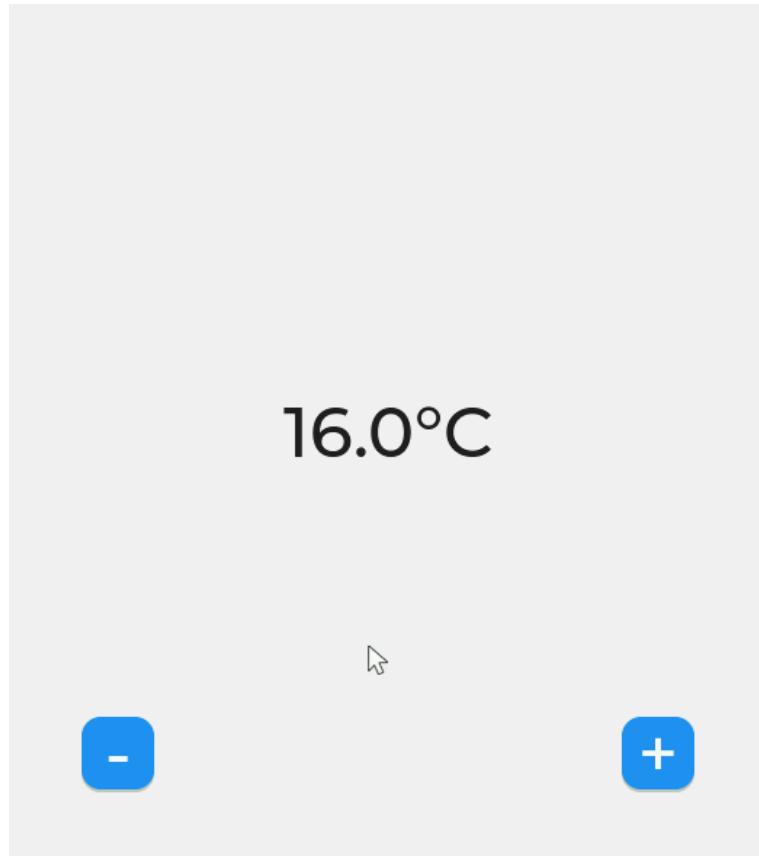


Figura 2 – Interfaccia grafica per il controllo della temperatura

2.1.4 Branches

Per organizzare efficientemente la repository sorgente, è 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` è 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.

2.2 pid-control

2.2.1 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 immagazzinare gli uuid dei sensori e poi legge effettivamente 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 effettuata 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 sacrificiamo un po di memoria per questo.

```

1  typedef struct
2  {
3      char id[16];
4      int16_t temperature;
5      uint8_t uint_id[OW_ROMCODE_SIZE];
6  } sensor;
7
8 // first we count the number of devices on the bus
9 while (diff != OW_LAST_DEVICE)
10 {
11     sensors_count++;
12     DS18X20_find_sensor(&diff, id);
13 }
14
15 // we malloc based on the number of sensors
16 sensor *sensors = malloc(sizeof(sensor) * sensors_count);
17
18 diff = OW_SEARCH_FIRST;
19 int i = 0;
20 // we store the sensor ids
21 while (diff != OW_LAST_DEVICE)
22 {
23     DS18X20_find_sensor(&diff, id);
24     sensor s;
25     for (int i = 0; i < OW_ROMCODE_SIZE; i++)
26     {
27         s.uint_id[i] = id[i];
28     }
29     sensors[i] = s;
30     i++;
31 }
32
33 while (1)
34 {
35     // now we read the sensor temperatures every second
36     if (DS18X20_start_meas(DS18X20_POWER_EXTERN, NULL) != DS18X20_OK)
37     {
38         fprintf(stdout, "error in starting measurement\n");
39         fflush(stdout);
40         delay_ms(100);
41         break;
42     }
43     for (int i = 0; i < sensors_count; i++)
44     {
45         sensor s = sensors[i];
46         if (DS18X20_read_decicelsius(s.uint_id, &temp_dc) != DS18X20_OK)
47         {
48             fprintf(stdout, "error in reading sensor %s\n", s);
49             fflush(stdout);
50             delay_ms(100);
51             continue;
52         }
53         fprintf(stdout, "sensor %s TEMP %3d.%01d C\n", s.id, temp_dc /
54         10, temp_dc > 0 ? temp_dc % 10 : -temp_dc % 10);
55         fflush(stdout);
56     }
57     delay_ms(1000);
}

```

2.2.2 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`.

2.2.3 Controllo pid

MONOTONIC CLOCK timer_fd

SCHEDULER PRIORITY

2.3 common-control

2.3.1 Comunicazione tra GUI e PID

Per fare in modo che l'interfaccia grafica e il processo di controllo pid comunichino è stato necessario comunicare attraverso dei file e utilizzando dei segnali.

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.

2.3.2 logging.h

Per stampare a schermo in modo ordinato i messaggi dell'applicazione è 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 può decidere a che livello filtrare i messaggi e anche se scrivere su file nel syslog o sulla console. Per evitare che più log si sovrascrivano a vicenda viene utilizzato un meccanismo di mutex. È stata aggiunta un'opzione per decidere la precisione del timer per intervalli di tempo sotto al secondo, utile per debuggare la regolarità del controllo pid.

2.3.3 Admin Control

2.3.4 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

```

1 AMEL_TEMP_CONTROL_VERSION = 44c17c6f2c492f1f3c7d8a6767df390c8d13eb9c
2 AMEL_TEMP_CONTROL_SITE = git@git.amelchem.com:mpapaccioli/temp-
control.git
3 AMEL_TEMP_CONTROL_SITE_METHOD = git
4
5 AMEL_TEMP_CONTROL_DEPENDENCIES = libevdev
6 AMEL_TEMP_CONTROL_GIT_SUBMODULES = YES
7
8 define AMEL_TEMP_CONTROL_BUILD_CMDS
9     cmake -DCMAKE_TOOLCHAIN_FILE=$(@D)/user_cross_compile_setup.cmake \
10         -B $(@D)/build -S $(@D)
11     make -C $(@D)/build -j
12
13 endef
14
15 define AMEL_TEMP_CONTROL_INSTALL_TARGET_CMDS
16     $(INSTALL) -d $(TARGET_DIR)/opt/amel-temp-control/
17     cp $(@D)/build/bin/lvglSim $(TARGET_DIR)/opt/amel-temp-control/main
18     cp -r $(@D)/build/lvgl/lib/* $(TARGET_DIR)/usr/lib
19
20 endef
21
22 $(eval $(generic-package))

```

Codice 5 – `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

```

1 AMEL_PID_VERSION = v0.0.3
2 AMEL_PID_SITE = git@git.amelchem.com:mpapaccioli/pid.git
3 AMEL_PID_SITE_METHOD = git
4
5 AMEL_PID_DEPENDENCIES = libmodbus
6
7 define AMEL_PID_BUILD_CMDS
8     make -C $(@D) CC=$(TARGET_CC)
9 endef
10
11 define AMEL_PID_INSTALL_TARGET_CMDS
12     $(INSTALL) -d $(TARGET_DIR)/opt/amel-pid/
13
14     cp $(@D)/pid $(TARGET_DIR)/opt/amel-pid/pid
15
16 endef
17
18 $(eval $(generic-package))

```

Codice 6 – 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 è stato aggiunto un ssh server per consentire il collegamento remoto al dispositivo embedded. È 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 è stata creata in modo da condividere l’accesso ad internet e dopo aver impostato come default gateway il pc creatore della rete bridge è possibile accedere alla rete globale dal sistema embedded. È stato utilizzato il dns server di google `8.8.8.8`. L’accesso ad internet è 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 è 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 è stato necessario installare un client 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 è stato abilitato di default il modulo cp210x, modificando il file `/drivers/usb/serial/Kconfig`.

Analogamente, è 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