



Università degli Studi di Milano

FACOLTÀ DI SCIENZE E TECNOLOGIE

Corso di Laurea Magistrale in Informatica

TESI DI LAUREA MAGISTRALE

Implementazione e sviluppo di una rete Mesh basata su dispositivi embedded general purpose

Candidato:

Alexjan Carraturo

Matricola **735952**

Relatore:

Andrea Trentini

Correlatore:

Elena Pagani

Anno Accademico 2014-2015

Indice

1 Introduzione	7
1.1 Analisi del lavoro svolto	8
1.1.1 Possibili destinatari	10
2 Le reti Mesh	11
2.1 Introduzione alle reti Mesh	11
2.1.1 La rete	11
2.1.2 Topologie di rete	12
2.1.3 La topologia Mesh	12
2.1.4 Algoritmi di routing	13
2.1.5 Better Approach To Mobile Ad-hoc Networking	14
2.2 Batman-adv	16
2.2.1 Caratteristiche di batman-adv	17
3 Hardware	21
3.1 Sistema Embedded	21
3.1.1 PLC	22
3.2 Microcontrollori	22
3.3 SoC	23
3.3.1 SoC, Licensing e Fab	24
3.4 Le architetture	26
3.4.1 RISC e CISC	26
3.4.2 ARM	27
3.4.3 MIPS	29
3.5 Esempi pratici	29
3.5.1 MCU, SoC e Board	29
3.5.2 Arduino	30
3.5.3 Discovery STM32	31
3.5.4 BeagleBone Black	33
3.6 Hardaware di prototipazione	35
3.6.1 La scelta del SoC	35
3.6.2 Hardware Utilizzato	36
4 Il sistema operativo	39
4.1 Introduzione al sistema operativo	39
4.1.1 Classificazione degli OS	39
4.2 I sistemi GNU/Linux	41
4.2.1 Storia dei sistemi GNU/Linux	41

4.2.2	Il kernel Linux	42
4.3	Root Filesystem	44
4.3.1	La libreria C	45
4.3.2	Init	45
4.3.3	La shell	47
4.4	Bootloader	48
4.4.1	U-boot	49
4.5	La fasi di boot	50
5	Implementazione	53
5.1	Implementazione Hardware	53
5.1.1	Form Factor	53
5.1.2	PCB	53
5.1.3	Costo	54
5.1.4	Hardware finale	54
5.2	Implementazione Software	55
5.2.1	Toolchain	55
5.2.2	L'ambiente di sviluppo	56
5.3	La scelta e compilazione del kernel	60
5.3.1	Vanilla, Mainline, Longterm e SoC version	60
5.3.2	Download ed ambiente di compilazione	62
5.3.3	Configurazione	63
5.3.4	Compilazione	66
5.4	Creazione del Rootfs	68
5.4.1	Buildroot Board Directory	68
5.4.2	Aggiunta di un pacchetto esterno	68
5.4.3	Configurazione Buildroot	70
5.4.4	Compilazione del rootfs	72
5.5	Dispositivo di memoria	73
5.5.1	Il partizionamento classico	73
5.5.2	La configurazione della scheda SD, e la copia dei file	76
5.6	uEnv.txt	77
5.7	mesh-init	78
6	MoM	81
6.1	Perché un nuovo protocollo	81
6.2	Il protocollo MoM	83
6.2.1	Il campo Tipo	84
6.2.2	Il campo Message	86
6.3	Implementazione software di MoM	88
6.3.1	libmom	88
6.3.2	MoM client	90
6.3.3	MoM server	90
7	Scenari di utilizzo	93
7.1	Automotive	93
7.2	Domotica	95
7.3	Sicurezza personale	96
7.4	Possibili sviluppi futuri	97

A Ambiente di test	99
A.1 Alternative per lo storage	99
A.1.1 NFS	100
A.1.2 TFTP	101
A.1.3 Utilizzo di NFS/TFT in U-Boot	101
A.2 Ambiente test della rete mesh	102
B Comandi Batman	103
B.1 Esempio di configurazione manuale	103
B.2 Un esempio di meshconf.txt	103
B.3 batctl	104
B.4 Alfred	106
B.4.1 Alfred-gpsd	106

Alexjan Carraturo

Capitolo 1

Introduzione

Quando la stampa è libera e ogni uomo in grado di leggere,
tutto è sicuro.

Thomas Jefferson, Presidente USA dal 1801 al 1809

Lo sviluppo tecnologico degli ultimi anni ha introdotto numerose possibilità di comunicazione, interazione e informazione. L'utilizzo di tecnologie mobili, di connettività a banda larga e di reti wireless è permeato completamente all'interno della vita dell'uomo medio. Nei paesi occidentali, la diffusione e la capillarità delle connettività alla rete internet ha raggiunto livelli raggardevoli.

Tali traguardi hanno portato con sé un numero crescente di preoccupazioni e interrogativi sulla condizione dell'essere umano all'interno dell'universo digitale. La percezione di internet, sia dal punto di vista personale che da quello infrastrutturale, è tuttora identificato come un'entità discesa dall'alto o come un prodotto concesso da grandi aziende ad aree ad alta rilevanza commerciale. Una simile visione è riscontrabile nello stridente paragone tra lo sviluppo di tecnologie IoT (Internet of Things) nelle aree economicamente evolute e la totale assenza di connettività in enormi aree geografiche per le cosiddette "zone depresse". Basti pensare che il coefficiente di penetrazione¹ di internet nei paesi africani è del 27% contro una media mondiale del 45%, con paesi come la Somalia intorno all'1.5% e l'Eritrea sul 1%. Anche l'Italia, se pur con dimensioni totalmente differenti, non è immune al Divario Digitale (dall'inglese Digital Divide): secondo recenti statistiche³, nel 2013 Internet è stato utilizzato dal 57,3% degli italiani, contro una media del 72% dei 28 stati membri dell'unione.

Un'osservazione più attenta mette in rilievo come, anche nei paesi occidentali, la popolazione sia effettivamente in prevalenza connessa ad internet, ma altresì evidenzia anche che le persone siano scarsamente connesse tra di loro. È possibile interagire con un conoscente che vive dall'altra parte del mondo, ma può risultare problematico entrare in contatto con le persone geograficamente più prossime.

La connettività, sia essa di ultima generazione o meno, è assai suscettibile ai capricci dell'ISP (Internet Service Provider), rendendo un guasto di un ripetitore, la rottura di un doppino o un problema di rete una macchina del tempo in

¹Calcolato in base percentuale sul numero di abitanti

²Fonte: internetworldstats.com

³Noi Italia, ISTAT

grado di riportare temporaneamente le persone in un'era di "pre-connettività". Sono le conseguenze di una centralizzazione dal punto di vista dell'accesso a internet e dei mezzi di comunicazione di nuova generazione (Social Network).

Una possibile alternativa al modello sin qui esposto è rappresentato dalle cosiddette reti "Mesh". Pur non entrando nei dettagli tecnici/implementativi che saranno trattati successivamente, è d'immediata comprensione quale sia la dimensione del cambio di paradigma. La connessione non avviene con un punto di accesso centralizzato, ma tra i singoli partecipanti alla rete che contestualmente sono produttori, propagatori e fruitori dell'informazione. L'idea è quella di una maglia (traduzione del termine "mesh") composta tra nodi, in cui i nodi stessi sono collegati ai prossimi in modo paritario, e dove tutti i nodi sono uniti tra di loro attraverso gli altri, formando qualcosa di più grande e resistente. L'idea di partecipazione e cooperazione di elementi paritari è aderente agli stessi principi alla base del software libero, un'importante chiave interpretativa ed elemento funzionale del progetto svolto.

Il campo di studi relativo alle reti mesh è tuttora molto attivo, e si stanno definendo nuovi ambiti applicativi. È in continuo fermento anche lo sviluppo di nuovi protocolli e standard, in modo da migliorare le possibilità d' inserimento nel contesto quotidiano del singolo. Alcuni di questi studi, come ad esempio quelli condotti in collaborazione tra il Computer Engineering and Networks Laboratory di Zurigo ed il Department of Electrical Engineering della Princeton University presentati nell'articolo "Routing Packets into Wireless Mesh Networks"[35], sono stati utilizzati come base teorica di riferimento e come prima ispirazione per questo lavoro.

Un altro importante settore di sviluppo tecnologico degli ultimi 20 anni è la categoria dei sistemi definiti "embedded" (testualmente: "incorporato"). Dentro questa definizione rientrano in realtà una notevole quantità di dispositivi, assai differenti per caratteristiche e scopo d'uso. Gli ambiti applicativi di questi sistemi possono essere le tecnologie mobili, fonia, domotica, telecomunicazioni, automotive, "infotainment", avionica, militare e biomedicale. Il miglioramento di questi sistemi in fatto di prestazioni, consumi, dimensioni e personalizzazione ha fatto sì che entrassero nell'esperienza di vita comune, come ad esempio negli smartphone, nel controllo remoto del riscaldamento domestico, nei sistemi di intrattenimento in auto ed aero e persino nei moderni televisori. Risulta evidente come buona parte del settore e dell'industria dei dispositivi embedded abbia avuto una notevole spinta in termini di sviluppo dal miglioramento delle tecnologie di connettività mobile, siano esse Wi-Fi, Bluetooth o ZigBee (802.15.4). Tale progresso simultaneo ha portato alla definizione di nuovi bisogni, inventando o reinventando interi settori di mercato.

1.1 Analisi del lavoro svolto

Ho creato un sistema e ho definito quali siano i parametri, i limiti e le differenze da considerare nello sviluppo di un sistema embedded sulla base delle specifiche iniziali. Dal punto di vista pratico, ho poi fatto in modo che la generazione del "sistema" (inteso come immagini kernel, rootfs, bootloader) sia il quanto più possibile automatizzata e infine ho descritto come eseguire delle customizzazioni successive.

Un'analisi schematizzata del lavoro suddivisa per passaggi:

- Definizione degli scenari di utilizzo
- Approfondimento sulle reti mesh
- Valutazione delle richieste dei vari scenari
- Definizione dell'hardware necessario (approfondendo accuratamente i connetti di embedded ed elencando le possibili alternative)
- Definizione del sistema operativo e delle sue componenti (approfondimento teorico e valutazione delle possibili alternative)
- Implementazione del sistema per l'hardware selezionato (argomentazione sulle possibili scelte implementative e termini di paragone con altri modelli di sviluppo)
- Realizzazione di un sistema di build automatizzato e semplificato
- Definizione di un protocollo di comunicazione specifico per gli scenari di utilizzo
- Implementazione software (ridotta) del protocollo
- Chiarificazione note tecniche comandi e ambiente di test

Per esigenze di natura espositiva, quesì passaggi non saranno presentati in questo ordine, ma in modo tale da costruire un'analisi progressiva, partendo dalle basi teoriche sino agli scenari di utilizzo. Gli obiettivi finali del lavoro svolto sono:

- Definizione di sistema hardware e software
- Valutazione dei criteri di scelta
- Automatizzazione build
- Implementazione di nuove soluzioni

L'idea generale di questa trattazione è quella di creare, nei limiti realizzativi, la definizione di un nuovo sistema, completo dal punto di vista hardware e software, esplicitandone i passaggi significativi. Una definizione di sistema deve partire dalla conoscenza dell'obiettivo da raggiungere, quindi dal tipo di dispositivo finale che si vuole ottenere e dalla destinazione d'uso al quale può essere rivolto. In questo caso si tratta di un dispositivo mobile, ideato per essere attivo per un periodo di tempo superiore alla giornata e collegato ad una rete WiFi, con opzionalmente anche l'ausilio di un dispositivo GPS. Questo tipo di richieste impone un'attenzione ai consumi complessivi. Un altro aspetto di rilievo dal punto di vista hardware è la riduzione dei costi del prodotto finale, valutando le componenti fondamentali da quelle non strettamente necessarie. Tale considerazioni si riflettono automaticamente nella prima fase di prototipo del software, che deve essere ideato e adattato per essere aderente al nuovo hardware, anche se non dovesse essere ancora fisicamente disponibile. Dal punto di vista del software, oltre a cercare di ottimizzare quanto più possibile sulla base delle considerazioni precedenti, è inoltre necessario produrre un processo che sia contestualmente facilmente ripetibile e modificabile, lasciando

ove possibile il maggior margine di manovra possibile alle eventuali evoluzioni future. Infine, è opportuno modellare il software in modo da semplificarne l'utilizzo e la configurazione da parte dell'utente finale. Parte integrante di questa tesi, oltre che realizzare il sistema finale funzionante, è mostrare tutti i passaggi di questo processo decisionale, basato per lo più considerazioni tecniche ed esperienze pregresse, definendo oltre al sistema anche un metodo lavorativo e decisionale. Particolare rilievo infatti è stato dato all'esame delle scelte di carattere tecnico e architettonico, ponendo l'attenzione sulle implicazioni pratiche, i termini di paragone, le diversità da altri modelli di sviluppo e le alternative disponibili, fornendo, ove possibile, un quadro completo ed esaustivo. Volendo sintetizzare questo tipo di analisi, si potrebbe dire che, al fine di fornire una maggiore comprensione collettiva, oltre che al cosa e al come siano state fatte le cose, è stata data particolare rilevanza ai "perché" alla base di determinate scelte. In tal senso, ho voluto riversare un parte delle mie esperienze professionali sia come Embedded Software Engineer sia come formatore per sistemi GNU/Linux enterprise.

Ignoranti quem portum petat nullus suus ventus est.

Nessun vento è favorevole per il marinaio che non sa a quale porto vuol approdare.

Lucio Anneo Seneca, lettera 71, Lettere a Lucilio

Una volta ottenuto l'insieme hardware/software funzionale allo scopo, verrà definito un nuovo formato di comunicazione e un'implementazione software di tale formato specificatamente ideata e sviluppata per questo progetto. Infine, saranno presentati alcuni scenari applicativi del dispositivo progettato e del formato di comunicazione; la possibilità di utilizzare le reti Mesh come metodo alternativo di collegamento ad Internet in aree metropolitane è stata già ampiamente discussa e presentata in numerosi "white paper" aziendali quali ad esempio, quello del colosso cinese HUAWEI[30] e quello dell'azienda americana di telecomunicazioni Aruba Networks[32](gruppo Hewlett Packard). Per tale motivo la trattazione si sposterà su ambiti applicativi differenti quali il domotico, la sicurezza personale ed automotive, senza tralasciare la possibilità offerta dalla progettazione iniziale di spostare l'attenzione anche sulla connettività ad internet ed altri progetti.

1.1.1 Possibili destinatari

Questo lavoro si rivolge a due possibili differenti destinatari: studenti ed imprese. L'esposizione che segue è stata ideata e pensata in modo tale da poter essere una sorta di manuale per quegli studenti che volessero affacciarsi al mondo dei sistemi embedded, cercando di esplorarne le possibili varianti. D'altro canto però, molte delle considerazioni tecniche e pratiche sul prodotto finito sono rivolte a coloro i quali si pongano come proposito di ideare, progettare e costruire un dispositivo embedded, conforntandosi con i più comuni ambiti applicativi del settore. Il mondo dei sistemi Linux Embedded è in grande crescita, richiede aziende mirate e professionisti pronti e qualificati: a tale proposito, questo testo può essere considerato una buona base di partenza per entrambe le tipologie di pubblico.

Capitolo 2

Le reti Mesh

2.1 Introduzione alle reti Mesh

2.1.1 La rete

Nel contesto informatico/telecomunicazioni, con il termine rete si tende ad indicare un insieme di nodi distinti (non necessariamente paritari) ed un insieme di collegamenti tra i vari nodi.

Per una maggiore comprensione di quanto verrà esposto successivamente, presentiamo rapidamente i due modelli (ISO/OSI e TCP/IP) utilizzati per definire uno stack (pila) di rete.

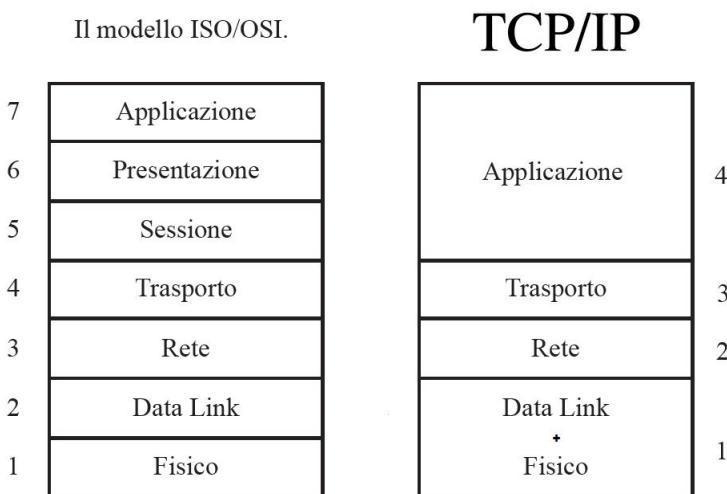


Figura 2.1: Comparazione della pila ISO/OSI e TCP/IP

Per semplificare molto si può dire che una comunicazione che parte da uno specifico nodo, attraversa tutti i livelli (layer) della pila partendo dall'alto e giungendo fino in fondo, per poi raggiungere attraverso un qualsivoglia mezzo fisico un nodo differente, ripercorrendo al contrario i livelli (dal basso verso l'alto) della pila. È particolarmente rilevante ai fini di questa trattazione evidenziare

come ogni passaggio tra un livello e il successivo altro abbia effettivamente un costo computazionale non sempre trascurabile.

2.1.2 Topologie di rete

Nel mondo delle telecomunicazioni vi è stata un'incessante evoluzione che è allo sviluppo di svariate tipologie di rete, distinguibili per vari fattori:

Estensione Geografica: sulla base della superficie coperta (es. BAN, PAN, LAN, CAN, MAN, WAN, GAN)

Canale Trasmissivo: il mezzo utilizzato per la comunicazione (es. Wireless, Cavo UTP, Linea Telefonica, Fibra Ottica, Satellitari, Reti Elettriche)

Topologia: la modalità in cui sono connessi i vari nodi (es. Linare, Anello, Albero, Stella, Bus, Maglia)

La distinzione topologica risulta essere particolarmente rilevante ai fini di questa trattazione. Si identificano le seguenti topologie.

Linare: (daisy-chain) Ogni nodo è collegato a due nodi adiacenti, tranne il primo e l'ultimo

Anello: (ring) Ogni nodo è collegato a due nodi adiacenti

Albero: (tree) Da ogni nodo possono partire catene lineari, partendo dalla radice (root) muovendosi sulle foglie (leaf)

Stella: tutti i nodi (spokes) sono collegati a un unico nodo centrale (hub)

Bus: tutti i nodi sono collegati a un unico bus di comunicazione

Maglia: (Mesh) Tutti i nodi trasmettono dati agli altri nodi.

2.1.3 La topologia Mesh

Una rete Mesh (dall'inglese "maglia") è costituita da nodi paritari detti "peers" non necessariamente fissi. Per quanto sia possibile implementare una rete mesh anche con dispositivi cablati, come ad esempio sfruttando collegamenti di tipo ethernet (Shorten Path Bridging, SPB IEEE 802.1aq), con il termine Rete Mesh si intendono per lo più reti con collegamento Wireless (Wireless Mesh Network, WMN), con particolare riferimento a nodi mobili e variabili. Dal punto di vista fisico una WMN può essere implementata con vari standard, come ad esempio IEEE 802.11 (WiFi), IEEE 802.15 (WPAN/Bluetooth) ed IEEE 802.16 (WiMax). Esiste inoltre lo standard IEEE 802.11s, estensione dello standard 802.11, atto a definire come i dispositivi wireless debbano essere collegati tra di loro per creare una rete mesh. I nodi, oltre ad inviare i propri dati, propagano dati ricevuti in ingresso destinati ad altri nodi; attraverso opportuni meccanismi di routing adattativo, tutti i nodi della rete sono raggiungibili con un numero congruo di passaggi attraverso nodi intermedi. Contrariamente a quanto potrebbe far pensare la dinamicità di questa topologia, le reti mesh sono in grado di offrire una notevole affidabilità, resistenza, resilienza e sono in grado di coprire vaste aree geografiche. Una rete mesh wireless può essere identificata

come una forma particolare della tipologia di rete Ad-Hoc. Una rete Ad-Hoc, o più specificatamente una Wireless Ad-Hoc Network (WANET) è una rete decentralizzata che usa meccanismi di “flooding” (testualmente innondazione) per la propagazione delle informazioni. Comunemente, questo tipo di attività viene svolta al cosiddetto “Layer 2” (Data Link Layer, Livello 2 del modello ISO/OSI), quindi, senza particolari accorgimenti non è possibile gestire attività di routing IP (layer 3, Network Layer). In questo schema il singolo nodo non conosce tutti gli elementi che compongono la rete, ma solamente gli elementi prossimi raggiungibili con un solo passaggio (1-hop). Sulla base di queste considerazioni risulta evidente come, non potendo rifarsi ad un livello fisico, siano necessari opportuni algoritmi di routing per poter gestire efficacemente le comunicazioni con tutti i nodi della rete, ottimizzando le rotte, anche nei casi in cui i nodi siano mobili e variabili.

2.1.4 Algoritmi di routing

Come anticipato in precedenza, risulta evidente che, per poter gestire efficacemente l’invio e la ricezione delle informazioni all’interno di una rete mesh, siano necessari, in particolare al crescere del numero dei nodi, algoritmi di routing efficaci e dinamici. Come accade spesso in settori di grande interesse e in pieno sviluppo, al problema del routing sono state proposte numerose soluzioni ed implementazioni: in questa sezione ne verrano esaminate alcune.

Link State Routing protocol

Il **Link State Routing protocol** è una delle due classi principali di algoritmi di routing utilizzato nelle reti packet switching (l’altra è nota come “distance vector”). Questo protocollo è implementato sui dispositivi atti a propagare i pacchetti all’interno della rete (switching node). Ogni nodo costruisce una sua mappa di connettività della rete stessa, organizzata con una struttura dati a grafo. Tale grafo mostra come i nodi siano collegati ad altri; ogni nodo è in grado di calcolare indipendentemente una rotta per raggiungere un qualsiasi altro nodo. In un primo momento, con un meccanismo noto come “*reachability protocol*” vengono scoperti i cosiddetti nodi vicini (detti anche “neighbours node”). Tale meccanismo viene eseguito in maniera periodica sul singolo nodo, ma in modo asincrono tra i vari nodi.

In un secondo momento ogni nodo invia le informazioni raccolte sui vicini (link-state advertisement), comprensive di un numero di sequenza in grado di identificare temporalmente queste informazioni. Tale informazioni vengono inviate a tutta la rete; questa fase prende il nome di “*flooding*” (trad. innondazione). In tal modo le informazioni raccolte, pur non essendo elaborate in modo sequenziale, possono essere disposte nel giusto ordine temporale.

Infine, una volta ricevuti tutti i “link-state advertisement” ogni nodo ricostruisce un proprio grafo rappresentante la mappa della rete.

Optimized Link State Routing Protocol (OLSR)

OLSR[22] è una versione del Link State Routing protocol pensata per le Mobile Ad-Hoc Network (MANET) ed è ottimizzato per reti di grandi dimensioni e dalla elevata densità. All’aumentare delle dimensioni e della densità della rete,

aumenta il livello di efficienza e ottimizzazione rispetto ad un classico algoritmo link-state. Essendo un algoritmo proattivo ha il vantaggio di poter ricalcolare dinamicamente le rotte e riduce sensibilmente il traffico durante la fase di “flooding”. Dal punto di vista pratico, ogni nodo sceglie un nodo a distanza minima stabilendo un collegamento bi-direzionale. Il nodo scelto diventa quindi un *Multipoint Relays (MPRs)* mentre il nodo che lo ha selezionato prende il nome di *MPRs Selector*. Il nodo MPRs invia periodicamente le informazioni di “Link State” agli altri nodi della rete: in tal modo il nodo annuncia alla rete la “raggiungibilità” dei nodi che lo hanno selezionato come MPRs. Anche in questo caso il “Control Message” dispone di un appropriato “Sequence number”, in modo da poter sequenziare l’arrivo dei messaggi di controllo mantenendo le informazioni temporali. Risulta inoltre possibile aumentare la reattività alle variazioni della rete riducendo l’intervallo di tempo che intercorre tra i vari messaggi di controllo.

Ad-hoc On-Demand Distance Vector (AODV)

AODV è un protocollo di routing pensato per le MANET e per altre tipologie di reti Ad-Hoc. AODV è un protocollo di tipo “reattivo”, ovvero il percorso di routing tra due nodi è calcolato solamente quando sia effettivamente richiesto da uno dei due, e rimane valido solo nell’intervallo di tempo in cui la “comunicazione” tra i nodi coinvolti è attiva. Dal punto di vista pratico la rete rimane “silenziosa” fino al momento in cui non sia richiesta una connessione. Nel momento in cui viene richiesta una connessione, parte una richiesta “broadcast” a tutti i nodi; i nodi inoltrano la richiesta che hanno ricevuto e registrano i mittenti delle richieste da loro ricevute sino al raggiungimento dei nodi coinvolti. A quel punto i nodi coinvolti dispongono di un grande numero di possibili rotte e sono in grado di scegliere quella che richieda il minor numero di passaggi intermedi (hop).

Questo approccio riduce al minimo l’overhead dovuto al protocollo di routing, ma aumenta notevolmente il tempo di attesa per il percorso di rete, aumentando la latenza complessiva.

2.1.5 Better Approach To Mobile Ad-hoc Networking

Batman (Better Approach To Mobile Ad-hoc Networking) è un protocollo di routing nato per sostituire OLSR[1].

Ogni nodo invia messaggi (Originator Messages, OGMs) broadcast per informare i nodi adiacenti della propria esistenza. A loro volta questi nodi reinoltrano gli OGMs ai loro vicini, notiziando questi ultimi sull’esistenza del nodo iniziale ed eseguendo così una sorta di flooding. Ogni OGM è dotato di un “Sequence Number”, impostato dall’originatore del messaggio, che permette ai nodi di poter riconoscere se questo determinato OGM è arrivato una o più volte; ogni nodo reinoltra al più un OGM, ovvero quello arrivato per primo. In tal modo il nodo riconosce come vicino più veloce rispetto all’originatore, quello che è stato in grado di inviare il messaggio più rapidamente. Questo meccanismo consente di poter costruire una tabella di routing efficace, con un basso overhead di rete (nel caso di routing layer3, il pacchetto “raw” costa circa 52bytes). In generale è errato riferirsi a B.A.T.M.A.N. come un unico algoritmo,

ma va considerato come un insieme di differenti algoritmi suddivisi per numero di versione progressiva.

Il modello di sistema

Modellando una rete come $G = (N, E)$, dove N rappresenta un insieme di nodi ed E rappresenta un insieme di link tra coppie di nodi, in B.A.T.M.A.N, per ogni nodo $i \in N$ esiste un insieme K di vicini a single-hop (distanza 1). Il messaggio trasmesso da una sorgente $s \in N$ verso una destinazione d è trasmesso attraverso un collegamento $(s, d) \in E$, se $d \in K$, altrimenti è trasmesso attraverso un cammino multi-hop (passaggio tra più nodi) composto da (s, i) e un cammino $[i, d]$ con $i \in K$ ed $(s, i) \in E$. Il cammino $[i, d]$ rappresenta il cammino tra il nodo i ed il nodo d attraverso la subnet $S = (N - s, A - (s, i) : i \in K)[8]$.

L'algoritmo B.A.T.M.A.N.

L'obiettivo dell'algoritmo è massimizzare la possibilità di consegnare il messaggio. BATMAN non controlla la qualità del Link ma solo l'esistenza. I link sono comparati valutando il numero di “originator message” ricevuti in un certo intervallo di tempo.

1. Considerare i messaggi di routing m da s a d nella rete G . Eliminare tutti i link $(s, i) \forall i \neq K$ per ridurre il grafo.
2. Associare per ogni link un peso w_{si} , dove w_{si} è il numero degli “originator message” ricevuto dalla destinazione attraverso i nodi vicini i all'interno di un certo intervallo di tempo.
3. Trovare il link con il peso w_{si} maggiore nel sottografo ed inviare m attraverso il link (s, i) .
4. Se $i \neq d$ ripetere i passaggi dall'1 al 4 per i messaggi di routing da i a d nel sottografo S .

B.A.T.M.A.N e non OLSR

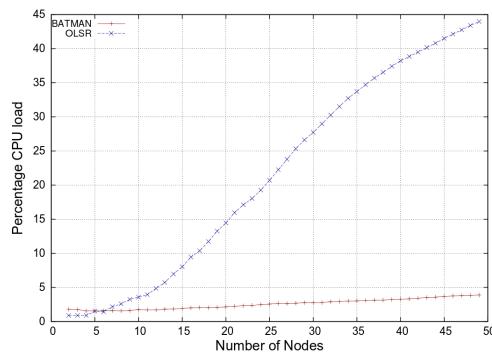


Figura 2.2: Uso della CPU tra BATMAN e OLSR all'aumentare del numero di nodi

Dall'analisi empirica[8] risulta che BATMAN offre performance migliori praticamente sotto ogni metrica di prestazioni. BATMAN offre un minor routing overhead in numero di bytes/sec. inviati dai singoli nodi: i risultati mostrano come BATMAN utilizzi 750Byte contro i 6000 utilizzati da OLSR per una mesh composta da 49 nodi. Anche dal punto di vista dell'uso della CPU (come visibile nella figura 2.2), Batman offre risultati migliori all'aumentare del numero di nodi, ciò lo rende preferibile nell'ottica di realizzare una rete caratterizzata da molti nodi su dispositivi con scarse performance computazionali e fortemente orientati al risparmio energetico.

2.2 Batman-adv

Batman-adv[14] (letto come batman advanced) è l'implementazione corrente di batman. In un primo momento B.A.T.M.A.N era realizzato con un demone¹ userspace di nome "batmand". Tale servizio lavorava al cosiddetto layer3 (detto anche livello IP), come una buona fetta degli algoritmi di routing per reti wireless. L'uso del layer3 implica che le decisioni sul routing vengano prese attraverso lo scambio di pacchetti UDP, e poi riversate nella cosiddetta "routing table" del kernel². **Batman-adv** opera interamente al layer2: ciò implica che sia il traffico dati, sia le informazioni necessarie al routing vengano spostate con delle ethernet frame definite "raw" (testualmente "grezze", ma nello specifico si intende non encapsulate). Tutte queste frame vengono gestite dal modulo kernel batman-adv, con un proprio encapsulamento in grado di emulare uno switch di rete virtuale tra tutti i nodi partecipanti: in pratica tutti i nodi simulano il comportamento di un collegamento locale, indipendentemente dalla topologia di rete.

Questo design offre alcune caratteristiche:

- Network Layer agnostico: si possono utilizzare protocolli differenti al di sopra di batman (IPv4, IPv6, DHCP, IPX...)
- I nodi possono partecipare alla mesh senza avere un indirizzo IP
- Facile integrazione dei client non mesh
- Roaming dei client non-mesh
- Flusso di dati ottimizzato attraverso la rete mesh

Per poter operare a layer2, è stato necessario spostarsi dallo spazio "userspace"³, a quello "kernel space"⁴. A tale scopo batman-adv è stato rilasciato come un modulo (vedere successivamente sezione relativa alla compilazione del kernel 5.3).

Tale scelta si giustifica considerando che il numero di letture/scritture necessarie sarebbe risultato insostenibile se eseguito da un demone userspace. Inoltre il demone avrebbe comunque dovuto interfacciarsi e scambiare dati con il kernel.

¹Servizio eseguito in background sul sistema.

²Nei kernel Linux le informazioni su IP e routing sono mantenute a livello kernel.

³Con il termine userspace si denota lo "spazio" in cui vengono eseguite le normali applicazioni

⁴Con il termine kernelspace si intende lo spazio di pertinenza esclusiva del kernel, con maggiori permessi e priorità sulla CPU

Lavorando direttamente da kernel space, il carico di lavoro dovuto alla gestione dei pacchetti diventa trascurabile, anche in caso di un traffico di rete intensivo.

2.2.1 Caratteristiche di batman-adv

AP Isolation

Come accade nei router commerciali è possibile, attraverso la **AP isolation**[12], impedire ai nodi collegati al medesimo BSSID⁵ di poter comunicare tra di loro. Nelle classiche strutture “Infrastructure”, il BSSID è rappresentato dal MAC Address dell’access point.

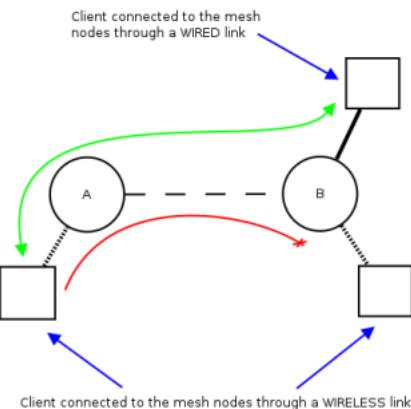


Figura 2.3: Uno scenario di utilizzo AP-isolation con BATMAN

Bridge Loop Avoidance

Batman-adv è in grado di riconoscere ed evitare i loop all’interno della rete mesh con il meccanismo del bridge_loop_avoidance[15]: un loop si crea quando più interfacce batX su differenti host vengono messe in “bridge” all’interno dello stesso segmento ethernet non appartenente all’insieme di BATMAN. Il loop generato è da intendersi solo per il traffico normale e non per il traffico generato da BATMAN. BATMAN ha un suo sistema per riconoscere i loop: I nodi definiti backbone (nodi BATMAN connessi alla stessa LAN) inviano uno speciale pacchetto per riconoscere gli altri nodi della LAN. Quando i nodi backbone ricevono un pacchetto mesh da un altro nodo backbone questo viene semplicemente scartato.

Distributed ARP Table

DAT (Distributed ARP Table)[16] è una cache ARP per la rete mesh che aiuta i client non mesh ad avere risposte ARP affidabili e con minore ritardo. In una comune rete mesh, ottenere un pacchetto broadcast (la richiesta ARP) dalla sorgente verso la destinazione (l’host con l’IP richiesto) richiederebbe svariate risstransmissioni dovute al fenomeno del “packet loss” (perdita dei pacchetti). DAT migliora la situazione creando una cache di richieste ARP relative a destinazioni note.

⁵basic service set identification

Bonding

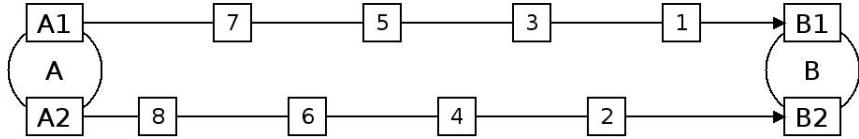


Figura 2.4: Invio dei pacchetti su link in modalità “bonding”

Come avviene per altre tipologie di connessione, è possibile applicare il meccanismo del “bonding”[18]. In sostanza si tratta di utilizzare un numero di interfacce maggiore di uno per il medesimo “data path”. I pacchetti sono inviati in modalità “round-robin”⁶. In teoria la velocità di trasmissione dovrebbe essere uguale alla sommatoria della velocità dei singoli link, ma ciò che è misurabile nella pratica è assai differente. In uno scenario di nodi collegati entrambi da 2 link in bonding, l’incremento non è uguale al doppio della velocità del singolo link, ma è all’incirca moltiplicato solo di un fattore 1,5. Le ragioni di tale comportamento sono ancora in fase di studio, e per questo motivo, questa modalità è considerata ancora sperimentale.

Fragmentation

Per poter trasmettere i pacchetti all’interno della rete, BATMAN aggiunge un particolare “header” (intestazione) a tutti i singoli pacchetti. Per fare ciò è necessario che il dispositivo di rete sia configurato in modo tale da avere un MTU⁷ maggiore di 1528 byte. In molti casi però le schede di rete cablate/wireless hanno un limite preimpostato a 1500.

Laddove non sia possibile aumentare il valore di MTU per limiti hardware o di driver, si applica il meccanismo della “Fragmentation”[17] (frammentazione), dove i pacchetti vengono spezzati e ricomposti nel processo di invio e ricezione. Tale meccanismo ha ancora delle forti limitazioni sul piano pratico ed in alcuni casi risulta inapplicabile.

Network Coding

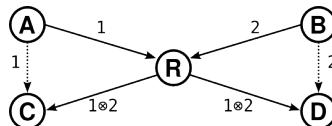


Figura 2.5: Scenario Network Coding

Il meccanismo del “Network Coding”[19] è un sistema ideato per poter ottimizzare i tempi di trasmissione, combinando due pacchetti in uno solo. La

⁶Tale termine, utilizzato solitamente per algoritmi di “schedule”, in questo contesto assume il valore di invio alternato di pacchetti su più link.

⁷MTU (Maximum Transmission Unit) indica la dimensione massima, in ottetti o byte della protocol data unit. In generale questo dato è associato a dispositivi di rete e, nei casi come ethernet o Wi-Fi può essere modificato.

creazione di simile combinazione richiede che il nodo che trasmette conosca entrambi i pacchetti, mentre il nodo ricevente deve conoscere almeno uno dei due pacchetti trasmessi per poter decodificare l'altro.

Un possibile scenario d'uso di questa tecnica è mostrato nella figura 2.5. Il nodo C e il nodo D ricevono rispettivamente il pacchetto 1 ed il pacchetto 2 dai nodi A e B. Il nodo R che ha ricevuto entrambi i pacchetti, invece di ritrasmetterli separatamente, attraverso il network coding ne trasmette soltanto uno combinato ai nodi C e D. Questi ultimi, avendo a loro volta ricevuto uno dei due pacchetti singolarmente, possono eseguire la decodifica.

Come mostrato dal grafico in figura 2.6, è possibile osservare che questo meccanismo offre dei vantaggi solo al superamento di una certa soglia di traffico dati. In caso di basso traffico, il vantaggio potrebbe risultare nullo o addirittura negativo, dovendo contemplare anche il tempo di computazione nella generazione dei pacchetti combinati.

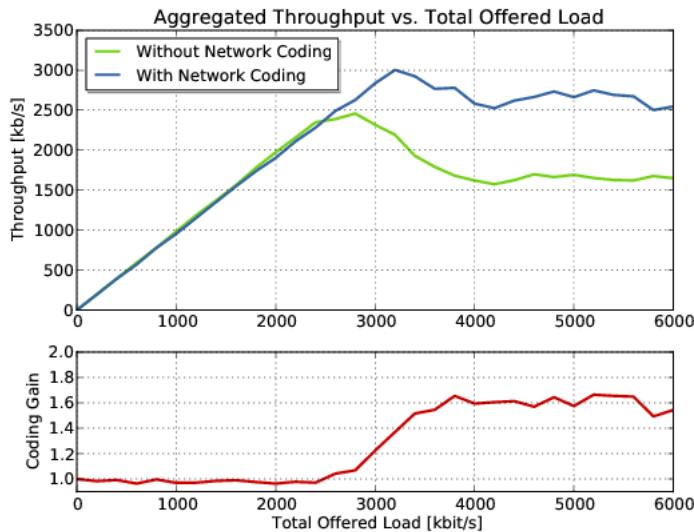


Figura 2.6: Performance del Network Coding

Alexjan Carraturo

Capitolo 3

Hardware

3.1 Sistema Embedded

Con la terminologia **Sistema Embedded** (tradotto testualmente “*sistema incorporato*”) si tende ad indicare l’insieme composto da hardware e software (occasionalmente definito firmware) dedicato a specifici scopi (“*specific purpose*”) i cui elementi siano tutti quanti integrati ed incorporati. Tale definizione può tuttavia risultare alquanto generica e fuorviante in quanto il numero dei dispositivi a rientrare sotto questa nomenclatura è elevato e composto da elementi assai diversificati per architettura hardware e software. Per comprendere meglio quale sia l’universalità dei sistemi embedded è possibile fornire alcuni esempi dei settori di utilizzo e sviluppo dei sistemi embedded:

- Avionica, Aeronautica, guida inerziale, sistemi di volo
- POS e Bancomat
- Stampanti e fotocopiatrici
- Elettrodomestici e Domotica
- router, switch, firewall
- Decoder, media player, telefoni, SmartTV
- Apparecchiature medicali

L’implementazione di questi dispositivi può richiedere differenti caratteristiche tecniche in base alle finalità per cui vengono progettati. La potenza computazionale di un sistema embedded è un fattore discriminante per identificare le varie categorie sulla base dell’architettura hardware necessaria. È possibile infatti fornire una prima suddivisione sulla base del tipo di elaboratore utilizzato in tre macro-gruppi:

- **PLC Programmable Logic Controller**
- **Microcontrollori**
- **SoC System-on-Chip**

3.1.1 PLC

I PLC (o Programmable Logic Controller) sono dei dispositivi relativamente semplici, generalmente pensati per lavorare su sistemi di automazione e non destinati alla produzione su vasta scala; in molti casi vengono programmati individualmente per svolgere una sola funzione specifica. Sono generalmente dotati di una buona connettività I/O per consentire la possibilità di interazione con sensori di vario genere (es. temperatura, pressione, peso e posizionamento) ed attuatori (es. cilindri idraulici, relays, motori elettrici o uscite analogiche). Un altro importante fattore dal punto di vista della progettazione di un PLC può essere legato alle condizioni di lavoro a cui deve essere sottoposto, quali ad esempio temperatura, umidità, polvere, stress meccanico e vibrazioni.

In base allo standard IEC 61131-3[21], i PLC sono programmabili con 4 linguaggi di programmazione, 2 testuali (ST e IL) e due linguaggi *grafici* (LD e FBD):

- **FBD** Function Block Diagram
- **LD** Ladder Diagram
- **ST** Structured text (simile al Pascal)
- **IL** Instruction List (simile all'assembly)

Per quanto i vari prodotti sul mercato condividano i concetti base di programmazione dei PLC, tali dispositivi non sono interscambiabili o compatibili tra di loro. Talvolta tale incompatibilità è presente anche tra dispositivi dello stesso produttore.

3.2 Microcontrollori

Analogamente allo sviluppo delle CPU general purpose, per intendersi quelle destinate al mercato dei computer portatili, desktop o workstation, già a partire degli anni 70 si sono distinti i primi **Microcontrollori** o MCU (Microcontroller unit). L'idea era quella di fornire un sistema hardware completo all'interno del singolo chip, tanto che in un primo momento questi dispositivi erano definiti come “Computer on Chip”. Storicamente, il primo elaboratore della categoria è stato l'Intel 8048 del 1975 con RAM e ROM all'interno del chip di elaborazione. Sin dai primi sviluppi, su alcuni modelli era possibile caricare del software adattato allo scopo designato su una apposita EPROM (programmata attraverso una finestra nel Chip attraverso la quale veniva mandato, tramite raggi UV, del linguaggio macchina). Tra gli anni '70 e gli anni '90 lo sviluppo dei microcontrollori è stato costante, aumentando quelle che erano le dotazioni disponibili. In particolare l'avvento delle EEPROM e successivamente delle Flash Memory, ha consentito un notevole miglioramento nei tempi di sviluppo e nelle possibilità di programmazione. Raramente su questi dispositivi, se non su quelli dell'ultima generazione, è possibile sfruttare dei sistemi operativi completi, sia per le ridotte performance computazionali, sia perché solitamente sono dotati di scarso spazio di memorizzazione. Per questo motivo molti produttori forniscono ambienti di sviluppo (sia ad alto sia a basso livello) specifici per le singole piattaforme, in grado di fornire gli strumenti fondamentali per la creazione di firmware ad-hoc.

Su molti microcontrollori moderni sono disponibili anche avanzati strumenti di debug e di scrittura della memoria interna (es JTAG). Ad oggi si parla di MCU caratterizzati da capacità di calcolo relativamente limitate (circa 200MHz nei modelli di punta, ma con una buona casistica sotto i 100MHz) se paragonate a quelle di CPU General Purpose (In alcuni casi sino a 4GHz) o dei SoC. Tale carenza dal punto di vista delle prestazioni è ripagata da ottimi valori di consumo energetico (assai inferiori ad 1W contro i 30-110W di una CPU General Purpose) e da eccezionali fattori di efficienza termica. Bisogna inoltre considerare che il prezzo per singola unità può essere assai inferiore se paragonato ad altre categorie di elaboratori, arrivando in alcuni casi a costare 50 volte meno di una CPU classica. Tali caratteristiche hanno determinato l'enorme successo dei MCU, che rappresentano buona parte del mercato dei sistemi embedded. Se pur non adatti ad applicazioni ad alta richiesta computazionale, sono eccellenti in quei sistemi embedded che svolgono operazioni semplici quali le misurazioni, la raccolta di dati ambientali, sistemi di controllo, regolatori ma anche in ambito industriale ed automotive. Inoltre, se utilizzati in prodotti soggetti ad economia di scala, il basso costo dei MCU li rende un'alternativa più interessante rispetto ai PLC, nonostante i maggiori costi di sviluppo. La dotazione di serie di un moderno MCU può essere molto ricca:

- Unità di elaborazione
- Memoria dati (RAM o EPROM)
- Oscillatore (Esterno o interno)
- Memoria programma (ROM, EPROM, FLASH)
- GPIO (General Purpose Input Output)
- Porte di comunicazione base (USART, I2S, SPI, I2C, USB)
- Porte analogiche (DAC, ADC, PWM)

Nei modelli più avanzati sono disponibili anche intefacce più complesse come Wifi, ZigBee, Ethernet, Touch Screen e LCD.

3.3 SoC

I **SoC** (acronimo di “System on Chip”) rappresentano l'ultimo stadio evolutivo dei sistemi embedded. Il termine, assai generico, è utilizzato per rappresentare un'ampia gamma di prodotti dalle maggiori potenze computazionali rispetto ai PLC e alle MCU e da dotazioni assai più ricche. La definizione di System on Chip deriva dal fatto che in molti casi si tratta di Chip che al loro interno contengono tutte (o buona parte) le componenti del sistema. A seguire alcuni esempi di componentistica reperibile all'interno un SoC;

- CPU (Core)
- GPU (Graphic Processing Unit)
- ISP
- Decoder/Encoder video

- Dispositivi di connettività (WiFi, Ethernet, Bluetooth, PAN...)
- Display Controller
- Audio Controller
- SPI, I2C, I2S, seriali, GPIO
- USB
- ADC/DAC
- Sensori (accelerometri, giroscopi, magnetometri, termometri...)
- GPS
- Modem
- Memoria RAM (sporadicamente)
- Memoria NAND/Flash (molto raro)

Pur esistendo SoC assai ricchi dal punto di vista della dotazione, è alquanto improbabile che tutti gli elementi presenti nella lista siano contemporaneamente presenti al loro interno. In generale, in base al costo e allo scopo di elezione, è possibile trovare un sottoinsieme di questa componentistica.

I SoC sono utilizzati praticamente ovunque, sia in ambito automotive, militare, domotico, Set Top Box (lettori multimediali, televisori, sistemi di "infotainment" domestici e di viaggio), navigatori satellitari, computer portatili, e negli ultimi anni hanno trovato vastissima applicazione nel settore della telefonia mobile. In termini pratici, si può affermare che siano presenti praticamente in ogni aspetto della vita digitale dell'uomo moderno. Data l'enormità del campo applicativo in cui i SoC sono utilizzati, risulta quantomeno complesso riuscire a delinearne una specifica generale. Volendo fornire un primo criterio di suddivisione si può parlare di due grandi architetture di riferimento (ARM e MIPS) e di una grande quantità d'implementazioni di queste due architetture. Parlando di un prodotto finito, riferirsi genericamente ad un processore ARM (o MIPS) è per lo più un errore del linguaggio comune. Ciò è dovuto sia alla grande quantità di sotto famiglie di queste due architetture, sia perché, a differenza di quanto sia possibile vedere nel campo delle CPU general purpose, il modello di business di questi prodotti è assai differente (Si veda la sezione 3.3.1).

3.3.1 SoC, Licensing e Fab

Per esperienza comune si è portati a credere che i Chip, ed in particolare i SoC, siano progettati, sviluppati e costruiti da un unico produttore in ogni sua componente. La realtà nel mondo dei SoC è però assai differente e, nella quasi totalità dei casi, tale credenza risulta falsa.

In primo luogo è assai raro che tutte le componenti del SoC siano state progettate dalla medesima azienda; nella stragrande maggioranza dei casi i vari elementi del SoC (che all'infuori del Core sono comunemente definiti con il nome di "IP") sono stati selezionati sul mercato in base alle loro caratteristiche tecniche ed al loro collocamento sul mercato. Il prezzo assume un'importanza

vitale, ed anche una differenza apparentemente irrisoria (ad esempio uno scarto di 0.10\$) può risultare determinante.

Inoltre, parlare di un Chip ARM o MIPS, è in generale sbagliato, in quanto queste aziende si limitano a progettare il Core ed alcune IP e, al netto di alcune rare eccezioni, non producono i Chip. Una volta progettato il Core infatti, si limitano a vendere i progetti, il software, la documentazione e l'assistenza tecnica in licenza ai produttori dei SoC, che in tal modo potranno comporre il loro sistema finale, combinando le ulteriori IP in base alle varie esigenze. A loro volta, anche le IP possono essere vendute in licenza, fornendo un servizio simile a quello offerto da chi licenzia il Core, ma senza effettivamente produrre fisicamente il componente in questione. Infine, una volta che un'azienda ha composto e progettato il SoC, unendo il Core e le singole IP, è alquanto raro che sia la medesima azienda a trasformare tale progetto nel prodotto finale. A parte alcune eccezioni eccellenti, la stragrande maggioranza delle aziende non dispone della cosiddetta "Fab"¹. Tali aziende, definite "Fabless" (senza Fab) si rivolgono a produttori terzi.

Nel paragrafo precedente abbiamo parlato genericamente di Licensing, senza specificare però che possono essere di due tipi distinti:

Licenza Core : vengono licenziati i progetti e gli strumenti per integrare il core (o la IP) all'interno del proprio SoC, ma senza il permesso di apportare modifiche di nessun tipo al core stesso.

Licenza Architetturale : vengono forniti gli strumenti e la possibilità di apportare modifiche all'interno del core. In generale assai più costoso sia dal punto di vista delle licenze sia dai costi di sviluppo.

In base a quanto visto sin ora, ed in considerazione del fatto che sia le IP che il Core possano essere “uniti” per possibilità ed esigenze ingegneristiche in modi differenti, è facile comprendere quale possa essere l'estrema variabilità strutturale dei vari SoC: facendo le dovute eccezioni, è alquanto improbabile una compatibilità totale dal punto di vista software, in particolare se si fa riferimento a implementazioni di “basso livello”². Questo modello di business ha permesso un rapido sviluppo di queste tecnologie, oltre che offrire un'estrema flessibilità progettuale, fondamentale per venire incontro alle esigenze dei singoli produttori. La possibilità di rivolgersi a “Fab” esterne ha di gran lunga facilitato l'ingresso sul mercato di numerose aziende che altrimenti non avrebbero potuto permettersi di produrre in proprio³. Inoltre, la possibilità di poter comprare dei “progetti” e dei “layer software” già pronti per essere adattati, ha ridotto di gran lunga quelli che erano i famigerati “Time to Silicon”⁴ ed il “Time to Market”⁵. Bisogna inoltre considerare che come avviene nel mercato dei Tablet e degli Smartphone, il SoC, a sua volta, potrebbe essere rivenduto ad un produttore terzo, per essere installato su un dispositivo completo, giungendo così ad un prodotto finale. L'estrema flessibilità ingegneristica e le forti esigenze di

¹Una “Fab” in questo contesto è il luogo dove vengono stampati i Chip.

²Tale termine non denota la qualità del software, bensì la vicinanza al linguaggio macchina o a codice kernel.

³Lo sviluppo e il mantenimento della produzione del silicio di nuova generazione ha dei costi elevatissimi difficilmente sostenibili da piccole e medie imprese.

⁴Il tempo per arrivare dal progetto al silicio

⁵Il tempo complessivo che intercorre tra l'inizio della progettazione e l'arrivo del prodotto finito sul mercato

ottimizzazione hanno avuto come conseguenza diretta lo sviluppo di software Open Source, ed in particolare dei sistemi GNU/Linux (o derivati), tanto da rendere questi sistemi le piattaforme di riferimento e sviluppo.

3.4 Le architetture

Nel corso dei numerosi anni della storia dello sviluppo dei sistemi embedded, sono state numerose le architetture proposte e utilizzate nei più svariati ambiti d'applicazione con alterne fortune. Per maggiore semplicità verranno prese in esame le due architetture che hanno riscontrato i maggiori risultati sul piano della presenza sul mercato.

3.4.1 RISC e CISC

Nella definizione dell'“instruction set architecture” (ISA) è possibile distinguere tra due grandi macro categorie che hanno caratterizzato lo sviluppo e le scelte architetturali delle CPU: RISC e CISC. Prima di approfondire tali concetti è giusto specificare che pur trattandosi di scelte apparentemente opposte, la linea di demarcazione tra queste diventa meno netta sul piano implementativo, in cui è possibile vedere delle architetture con ISA “ibride”. In questi termini è più corretto parlare di “Approccio RISC o CISC”.

CISC

CISC (Complex instruction set computing) è un termine creato a posteriori rispetto allo sviluppo di queste architetture (per denominare l'approccio opposto a RISC). Con tale dicitura si intende quella tipologia di processori caratterizzati da architetture con un notevole numero di “low-level instructions” (istruzioni a basso livello) ed in cui le istruzioni sono a lunghezza variabile e possono richiedere molteplici cicli di clock per essere eseguite.

Il loro vantaggio è offrire un linguaggio macchina con istruzioni non molto distanti da quelle di un linguaggio di alto livello, quindi apparentemente più facili da programmare. Contestualmente però, tale approccio risulta più costoso in termini di “silicio”, ovvero come numero di componenti interne necessarie a realizzare l'architettura. Alcune esempi di CPU che nel tempo sono stati etichettati come CISC:

- System/360
- VAX
- PDP11
- Z80
- Motorola 68000
- Intel 8080
- MT6502
- x86

L'architettura x86 è un esempio di quanto il confine tra questi due approcci possa diventare sottile in fase realizzativa; se apparentemente si presenta con le caratteristiche tipiche del CISC, con un elevato numero di istruzioni, in realtà dispone di una sezione interna atta alla traduzione delle istruzioni CISC in microistruzioni, elaborate successivamente in modo simile a quanto avviene nei processori RISC.

RISC

RISC (Reduced Instruction Set Computing) è un design per CPU basato su un “instruction set” semplificato, composto da istruzioni semplici, molto ottimizzate e in grado di essere eseguite con pochi cicli di CPU. Il vantaggio di un simile approccio è di avere un basso costo in “silicio” e una grande efficienza, ma a costo di una notevole distanza dai linguaggi di alto livello e con una conseguente elevata difficoltà di programmazione a basso livello. Per colmare tale distanza sono necessari dei compilatori assai più efficienti rispetto al modello CISC. Alcuni esempi di processori RISC:

- ARM
- ARC
- Blackfin
- Atmel AVR
- MIPS
- PA-RISC
- SuperH
- SPARC

A maggior dimostrazione di quanto detto in merito al sottile confine tra CISC e RISC sul piano pratico, c'è da considerare il fatto che ad oggi molti processori dichiaratamente RISC, hanno un instruction set più ricco di alcuni processori CISC. Il considerare il termine “Reduced” riferito al numero di istruzioni potrebbe infatti risultare fuorviante, e sarebbe forse più indicato considerarlo in merito alla complessità delle istruzioni.

Osservando la lista delle architetture RISC, è facile notare come la maggior parte di quelle più affermate nel mondo embedded appartenga a questa famiglia, mentre quelle CISC abbiano dominato in ambito Desktop/Server/Mainframe per lungo tempo.

3.4.2 ARM

I primi processori ARM nascono dalla Acorn Computers (da lì il primo significato dell'acronimo: Acorn Risc Machine) nei primi anni 80. In un'epoca in cui si scorgeva già il futuro dominio dei computer definiti “IBM compatibile”, l'azienda di Cambridge aveva rivolto la propria attenzione ad un'implementazione migliorata del già noto MT6502 (considerato troppo modesto per adeguarsi alla nuova era delle interfacce grafiche). L'idea generale era di produrre un'architettura

estremamente semplice ed efficiente, da contrapporre alla complessità delle architetture Intel e Motorola, riducendo significativamente i consumi. Se in un primo momento la Acorn aveva scelto di produrre direttamente i Chip, appoggiandosi a VLSI come Silicon Partner, in un secondo momento decise che sarebbe stato più conveniente licenziare i propri prodotti a ditte terze.

Nel corso degli anni, ARM (divenuta poi Advanced Risc Machine Ltd) ha proseguito lo sviluppo, raggiungendo la leadership in tutti i campi del settore embedded, ed arrivando di recente ad aggredire il mercato home computing e server.

Architettura e Core

Occore fare alcune precisazioni relative alla nomenclatura dei core e delle architetture:

Architettura: con architettura si indica il macroinsieme di CPU che condividono il medesimo ISA (Instruction Set Architecture), al netto delle estensioni. Ad una singola architettura possono corrispondere dei Core assai differenti.

Profilo: a partire dalla famiglia dei Cortex, si identificano tre diversi profili sulla base dello scopo per cui è stato progettato il core;

- Profilo A (Application): in genere utilizzati per dispositivi multimediali quali tablet, smartphone, set top box.
- Profilo R (RealTime): utilizzati per sviluppare dispositivi in grado di lavorare in tempo reale. Il requisito real-time richiede numerose interventi sia hardware che software.
- Profilo M (Microcontroller): sono utilizzati per produrre microcontrollori (MCU).

Core: il Core identifica con precisione l'architettura, le estensioni architetturali utilizzate e le caratteristiche interne.

A seguire una tabella di sintesi su alcune delle architetture e dei core di riferimento. La lista completa è assai più lunga ed esaustiva (per non parlare degli esempi), ma non utile ai fini della trattazione.

Architettura	Profilo	Core	Esempio
ARMv1	-	ARM1	-
ARMv2	-	ARM2,ARM250, ARM3	-
ARMv3	-	ARM6, ARM7	-
ARMv4	-	ARM8	StrongARM
ARMv4T	-	ARM7TDMI, ARM9TDMI	XScale
ARMv5	-	ARM7EJ,ARM9E,ARM10E	TI DaVinci
ARMv6	-	ARM11	Raspberry Pi
ARMv6-M	M	Cortex-M0(+), Cortex-M1	Arduino ZeroPro
ARMv7A	A	Cortex-A(5,7,8,9,12,15,17)	BeagleBoneBlack
ARMv7R	R	Cortex-R(4,5,7) -	-
ARMv7M	M	Cortex-M(4-7),	STM32(F4xx)
ARMv8A	A	Cortex-A(53,57,72)	Apple iPad

3.4.3 MIPS

MIPS (acronimo di Microprocessor without Interlocked Pipeline Stages) nasce come progetto di sviluppo presso la Stanford University (Santa Clara, California) nei primi anni 80. La particolarità, da cui deriva il nome, è che tutte le istruzioni presenti nella pipeline dovevano essere concluse in un solo ciclo di clock, evitando stalli e ritardi ed eliminando di fatto la necessità di un sistema di controllo (interlock). L'idea alla base delle architetture MIPS (e in generale delle architetture RISC) era semplificare le operazioni complesse (es. moltiplicazioni e divisioni) in una serie di operazioni più semplici. Tale approccio, se pur accademicamente interessante, fu poi abbandonato nel momento in cui la neonata MIPS Computer Systems (poi MIPS Technologies dopo l'aquisizione di SGI) si propose nel mercato. A partire dagli anni '90 i processori MIPS furono licenziati a produttori terzi, e furono proposte architetture a 32bit (MIPS32) e a 64bit (MIPS64). A differenza di ARM, MIPS trova maggiore successo nel mercato dei SuperComputer, ovvero per quelle macchine progettate per disporre di una grande potenza di calcolo. Tra i core ideati da MIPS, risultano di grande successo il MIPS R3000A (PlayStation) e R4000 (Playstation 2).

Dal 2012 MIPS Technologies è passata sotto il controllo di Imagination Technologies, e sono stati annunciati nuovi prodotti appartenenti ai profili M (Microcontroller, paragonabili alla famiglia dei CortexMx), I (Interaptiv, paragonabili alla famiglia dei CortexRx e con la fascia bassa dei CortexAx) e P (Performance, paragonabili alla famiglia dei CortexAx di fascia alta).

3.5 Esempi pratici

3.5.1 MCU, SoC e Board

In generale, sia per quanto riguarda i SoC che per quanto riguarda le MCU, si tende ad integrare più componenti possibili all'interno del Chip. Ciò nonostante il solo chip, per quanto funzionale così come prodotto, sarebbe difficilmente utilizzabile per prototipare e sviluppare il prodotto finale. Oltre alla comune gestione energetica (evidentemente necessaria) molti dispositivi necessitano di spazio di storage o di esportare alcune porte di I/O in modo semplice da interfacciare, o ancora di sensoristica accessoria.

Per questi motivi, molti produttori di chip di varia grandezza e potenza, oltre a fornire la documentazione e talvolta gli strumenti per lo sviluppo software, mettono a disposizione dei clienti anche delle board che a seconda dei casi prendono il nome di “*evaluation board*” o “*development board*”.

Una **eval board**, è solitamente realizzata su PCB di piccole/medie dimensioni, è ideata per far facilitare lo sviluppo hardware e software di prodotti basati sul chip di riferimento, oltre che per promuovere il chip stesso presso gli sviluppatori o i produttori di dispositivi. In alcuni casi infatti, soprattutto in ambienti marketing, queste board assumono il nome di **demo board** (“*demonstration board*”).

Senza scendere troppo nel dettaglio, una *eval board* dispone generalmente di tutti o quasi i collegamenti fisici per l'hardware presente nel MCU o nel SoC, oltre che ad una serie di dispositivi accessori atti a dimostrarne tutte le potenzialità in ambiti applicativi differenti. Oltre ai collegamenti classici (rete, video, audio, bus), una caratteristica mediamente molto apprezzata dagli

sviluppatori è la disponibilità di connessioni per il debug a basso livello (es. JTAG), raramente presenti sul prodotto finito.

In base a queste considerazioni si può facilmente capire come la presenza di una eval board ben progettata possa concorrere pesantemente al successo o meno di una determinata piattaforma. In alcuni casi, il successo della eval board ha superato quello della piattaforma stessa, facendola considerare come un prodotto finito da utilizzare così come è, e non per la finalizzazione di un dispositivo differente.

Validation Board

Un caso particolare sono le cosiddette “**Validation Board**”; si tratta di norma di board dalle grandi dimensioni usate per validare una piattaforma (MCU o SoC). Sono di grandi dimensioni perché sono spesso dotate di numerose porte di debug, batterie di pin e test point. Lo scopo di queste board è in genere di effettuare la “silicon validation”, ovvero un processo di test e verifica atto a evidenziare la presenza o meno di errori di progettazione o di lacune strutturali all’interno del Chip. Raramente una validation board esce sul mercato, ed in generale sono create di volta in volta e in numero assai limitato.

3.5.2 Arduino

Nato ad Ivrea intorno al 2005[2], Arduino nasce in Italia come progetto di prototipazione elettronica a basso costo per fini didattici. Dedicato al mercato hobbistico ed amatoriale, ha tra le sue caratteristiche più apprezzate la possibilità di disporre completamente di tutti i progetti hardware (viene definito Open Hardware) oltre che la presenza sul mercato di Kit che consentono di personalizzare la board di sviluppo (o di assemblarla interamente).

La piattaforma arduino ha avuto uno sviluppo rapidissimo in termini di evoluzione con numerose implementazioni differenziate tra di loro a partire dalla scelta del MCU.

Alcuni modelli:

Serial Arduino ATmega8, porta seriale

Arduino Extreme ATmega8, porta USB

Arduino Mini ATmega168 SMD, board mini

Arduino Nano ATmega168 SMD, board nano, USB

LilyPad Arduino ATmega168 SMD, wearable

NG ATMega8, porta USB

NG Plus ATMega 168, porta USB

BT ATmega168, Bluetooth

Diecimila ATMega168, porta USB

Duemilanove Atmega328, Alimentazione DC

Mega ATmega1280, memoria addizionale.

Mega2560 ATMega2560

Due Atmel SAM3X8E Cortex-M3

Zero Pro Atmel SAMD21 Cortex-M0+

Per comprendere le differenze tra le varie piattaforme si possono comparare l'ATMega8[4] e il SAM3X8E[5]

	ATMega8	SAM3X8E
arch	AVR	ARM
bit	8	32
flash(kb)	8	512
Max I/O Pin	23	103
Max freq(MHz)	16	84
USB	-	1

Sviluppo software

Tra i motivi del grande successo di Arduino nel campo hobbistico, oltre al fatto di essere molto aperto ed economico, c'è anche l'ambiente di sviluppo. Le piattaforme Arduino infatti dispongono di un ambiente di sviluppo (IDE) derivato da Wiring (ambiente di sviluppo integrato scritto in Java e rilasciato con licenza open source) e basato su Processing (linguaggio di programmazione ad oggetti simile a Java e rilasciato con licenza GPL). Sia il linguaggio di programmazione che l'IDE sono pensati e sviluppati al fine di risultare estremamente semplici dal punto di vista del programmatore, mascherando il più possibile gli aspetti legati al lavoro a basso livello, per concentrarsi sugli aspettivi creativi e funzionali. Ciò ha portato a una maggiore accessibilità allo sviluppo anche per coloro che non avessero grande dimestichezza con la programmazione. I programmi realizzati con questo IDE prendono il nome di “sketch”. La programmazione del dispositivo con lo sketch è piuttosto semplice, e differisce tra i vari modelli per le caratteristiche di collegamento del singolo dispositivo (USB, Seriale, Bluetooth).

3.5.3 Discovery STM32

STM32 è una famiglia di microcontrollori creati da STMicroelectronics, basati sui core ARM, profilo M. L'estrema versatilità, l'ampia gamma di prodotti e la semplicità di sviluppo su questa piattaforma, hanno attirato l'interesse di numerosi sviluppatori, oltre che una discreta adozione all'interno di progetti di sistemi embedded.

Anche in questo caso non è possibile fare riferimento ad STM32 come ad un unico Chip; nel corso degli anni STMicroelectronics ha fornito svariati modelli di questo MCU, differenziati notevolmente da hardware e dotazione di base.

Famiglia	Core	SRAM	Flash	Velocità	Novità
F0	Cortex-M0+	4-20 Kb	16-128Kb	48MHz	-
L0	Cortex-M0+	8Kb	32-64Kb	32MHz	USB
F1,F2	Cortex-M3	4-128Kb	16-1024Kb	72-120MHz	F2(SD, Ethernet, USB Otg)
F3,F4	Cortex-M4F	16-192Kb	64-2048Kb	72-180MHz	F4(LCD-TFT)
F7	Cortex-M7	-	-	-	-



Figura 3.1: Una discovery board STM32 F429 dotata di LCD

Una board di recente produzione basata su STM32 è la 32F429IDiscovery[31]. A seguire le caratteristiche di questa eval board:

- Core: Cortex-M4F 84/168/180 MHz
- Static RAM: 512/1024/2048Kb
- USB 2.0 OTG
- CAN 2.0B
- SPI/I2C/I2S, USART, UART, SDIO, ADC, DAC, GPIO, WDT
- LCD-TFT controller

Software

Nonostante ci siano stati già dei risultati incoraggianti dal punto di vista del porting di sistemi GNU/Linux su questo tipo di piattaforma, le prestazioni ridotte (se comparate con un SoC) non rendono questa alternativa tuttora valida. In alternativa, esistono un gran numero di ambienti di sviluppo (Keil, IAR, Atollic, TrueStudio o anche direttamente una toolchain basata gcc) che consentono di sviluppare “from scratch”⁶ una propria applicazione personalizzata, appoggiandosi o meno a pseudo sistemi operativi real-time (es. FreeRTOS).

Date le ridotte dimensioni della SRAM e della Flash, si predilige per lo più questa seconda alternativa, massimizzando l’efficienza computazionale del device, ed eseguendo solamente il codice strettamente necessario.

⁶In questo contesto si riferisce alla possibilità di partire solo dal codice sorgente con solo a disposizione il BSP.

Ambito applicativo

La famiglia di MCU STM32, così come alcuni dei prodotti dei principali concorrenti (serie MSP di Texas Instruments, Kinetis di Freescale) trovano ampio utilizzo in settori come il comparto automotive, il settore biomedicale e l'ambito militare. A differenza di quanto avviene in altri settori, la scelta di usare una MCU al posto di un SoC non è legata strettamente a semplici fattori di costo (che pur influenzano in fase di progettazione) e talvolta nemmeno legata a fattori di consumo energetico.

L'estrema semplicità architetturale, l'elevato grado di personalizzazione e la possibilità di poter lavorare senza particolari problemi con sistemi OS-less⁷, consentono alle aziende di poter sviluppare hardware e software in grado di superare i rigidi protocolli di certificazione a cui sono sottoposti i prodotti delle sopraccitate categorie (automotive, biomedicale, militare). In molti casi infatti si tratta di sistemi che non devono svolgere gravosi compiti computazionali, ma che devono essere certificati e testati in modo tale da fornire la massima affidabilità: nel settore biomedicale, in particolare per apparecchi che devono operare per il monitoraggio dei parametri vitali dei pazienti, ogni singola riga di codice deve seguire delle regole specifiche di programmazione, interazione e documentazione[27]. Ottenere un risultato analogo con software “Linux oriented” rappresenterebbe uno sforzo mastodontico, antieconomico e poco produttivo.

3.5.4 BeagleBone Black

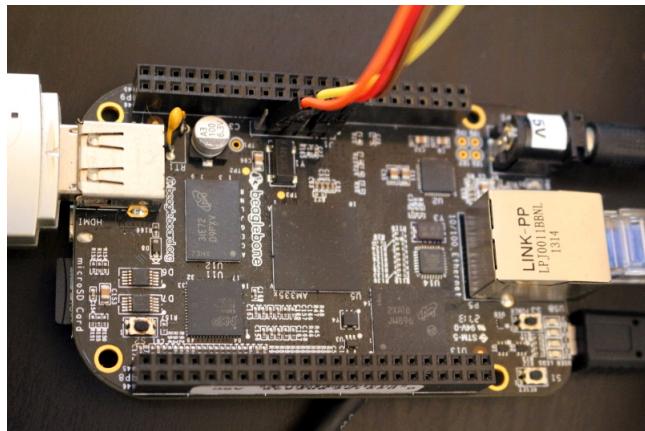


Figura 3.2: Una BeagleBone Black collegata

Ricordando quanto già evidenziato in precedenza, una eval board deve essere un buon compromesso tra flessibilità di utilizzo, semplicità di programmazione e debug, mantenendo la possibilità di mostrare tutte le capacità del SoC. Tutto ciò che si trova al di fuori del SoC è aggiunto per soli motivi pratici e dimostrativi.

Caratteristiche:

- Processore: AM335x 1GHz ARM® Cortex-A8
- 512MB DDR3 RAM

⁷Privo di un sistema operativo completo.

- 4GB 8-bit eMMC on-board flash storage
- USB client
- USB host
- Ethernet
- HDMI
- 2x 46 pin headers

TI AM335x Sitara

Specifiche tecniche[24]

- CPU: Arm Cortex-A8 32 Bit RISC 1Ghz
- NEONTMSIMD Coprocessor
- L1 Cache (32 Kb Data, 32 Kb Instruction) Parity single error
- L2 Cache 256KB ECC
- 176KB BootROM
- JTAG, cJTAG (Boundary Scan, IEEE1500)
- mDDR2,DDR2,DDR3 (200-800MHz)
- PRCM Module
- RTC
- CryptoHardware Accelerators (AES, SHA, PKA, RNG)
- PowerVR SGX530
- General Purpose Memory Controller (8/16 bit)
- 2 High Speed USB2 OTG port
- 2 Gigabit MAC Ethernet (10/100/1000Mbit) MII, RGMII, RMII, MDIO
- 6 UART (1 full-modem)
- 3 MMC,SDIO,SD port
- 3 I2c, 2 McSPI, 128 GPIO, 1 ADC (12Bit, 200K samples), 3 PWM
- LCD Controller (2048x2048)
- Package 324Pin

Consumi[25]:

- VMIN: 0.95V a 300MHz
- VMAX: 1.325V a 1GHz

- Potenza a 600MHz in mW:

min 553.9 (LinuxPSP)

max 823.21 (3D)

- Potenza a 1GHz in mW:

min: 780.54 (Idle)

max: 1142.02 (Drystone)

I consumi sono mostrati sulla base di specifici casi d'uso: Idle e LinuxPSP rappresentano casi a bassa richiesta computazionale, Drystone e 3D invece sono test per valutare il massimo delle performance.

Configurazioni alternative

Il SoC consente effettivamente di poter utilizzare tutte le specifiche elencate sopra, ma non contemporaneamente. Molti pin di I/O del SoC sono “multiplexed”, ovvero sono condivisi tra più possibili configurazioni. Se ad esempio si usa l’LCD controller, l’Ethernet ed un MMC il numero di GPIO disponibile sarà sensibilmente ridotto. In questo, come in molti SoC, esistono quelle che vengono chiamate le “Alternative Configuration” che indicano l’uso che viene fatto dei singoli I/O in base alla configurazione scelta.

Apparentemente una simile scelta potrebbe risultare illogica, ma bisogna sempre ricordarsi che questo tipo di chip è pensato per rispondere ad esigenze embedded, quindi con dei forti limiti dal punto di vista dei consumi ma anche delle dimensioni stesse del chip. Inoltre, l’utilizzo di tutte le possibili porte di un SoC è alquanto raro; usando i pin multiplexed, si può scegliere la configurazione addata alle esigenze ingegneristiche, mantenendo basse le dimensioni di stampa e disabilitando ciò che non sia necessario. Tale scelta impatta positivamente sia sul consumo energetico, sia nella complessità di progettazione del PCB finale del prodotto.

In base alla versione del silicio, il Sitara può essere testato per diversi range di temperatura: commerciale (0-90C), industriale (-40C,90C), estesa (-40C,105C)[26]. Sempre in base alla versione del silicio e al distributore il prezzo può variare tra i 10 ed i 14 dollari al pezzo (per ordini superiori alle 1000 unità).

3.6 Hardware di prototipazione

3.6.1 La scelta del SoC

Nonostante le MCU abbiano raggiunto risultati eccellenti dal punto di vista del rapporto consumi/prestazioni, e che con alcuni pesanti accorgimenti sia possibile utilizzarle con sistemi operativi completi (versioni modificate del kernel Linux come uLinux), le non elevate prestazioni computazionali, la complessità di sviluppo (in particolar modo del supporto hardware) e la scarsa disponibilità di memoria di massa, renderebbero assai più complicata la realizzazione del dispositivo finale.

I minori consumi (60 mW, 187uA/MHz), la maggiore semplicità dal punto di vista del “layout” e il costo inferiore per singolo componente rendono la scelta di una MCU molto interessante, ma si dovrebbe lavorare con un sistema al

limite delle sue possibilità, soffrendo sia dal punto di vista prestazionale, che per quanto riguarda gli aspetti di sviluppo, debug, e di flessibilità complessiva di ambito applicativo.

La scelta quindi ricade su un SoC, in questo caso il Texas Instruments Sitara AM3358 (vedere sezione 3.5.4): il costo ed il consumo sono sicuramente superiori se paragonati ad un MCU, ed è decisamente sovradimensionato per le esigenze computazionali richieste dal progetto. Inoltre, pur disponendo di un core con ISA ARMv7, è un modello piuttosto datato. Va detto però che, a differenza di quanto avviene nel mondo Desktop, ciò può risultare vantaggioso sotto più aspetti: da un lato mantiene basso il costo per singola unità, dall'altro, il lungo periodo sul mercato ha consentito un elevato grado di “maturità del silicio”⁸, un fattore fondamentale per garantire l'affidabilità del prodotto finale. Inoltre, l'elevata adozione di questo SoC, fa sì che esista una buona base software di partenza disponibile.

3.6.2 Hardware Utilizzato

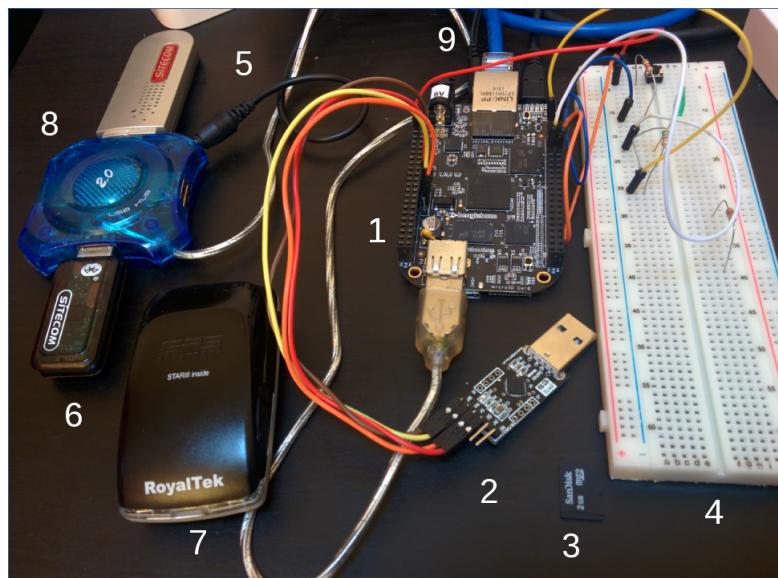


Figura 3.3: L'hardware di prototipazione utilizzato

Quello mostrato nella figura 3.3 è l'hardware utilizzato per la prototipazione, lo sviluppo ed il test del software. Si procede con un esame dettagliato.

1. BeagleBone Black (vedere sezione 3.5.4)
2. USB to Serial TTL cp102x: Convertitore da USB a porta seriale in TTL (5V).
3. Scheda SanDisk MicroSD da 2Gb

⁸Caratteristica delle CPU in generale che identifica attraverso precisi criteri il grado di testing, validazione, “silicon fix” ed eventualmente respin del silicio.

4. BreadBoard con alcune componenti montate

Bottone 4 pin

Led verde

Resistenza 820 Ohm

Resistenza 560 Ohm

5 Cavi di collegamento

5. USB Adapter WiFi 54G Sitecom WL-168v1 (chip RTL8187)

6. USB Adapter Bluetooth Sitecom CN 512 V1 002

7. GPS Bluetooth Royaltek StarIII

8. Generic USB Hub (alimentato 5v/1A Max)

9. Collegamento Ethernet

Alcuni degli elementi mostrati sono necessari esclusivamente nella fase di sviluppo e testing A.1, ma assolutamente non necessari nella parte finale. Ad esempio la porta seriale, il collegamento Ethernet e l'Hub USB non saranno presenti (vedere sezione 5.1.4).

Schema

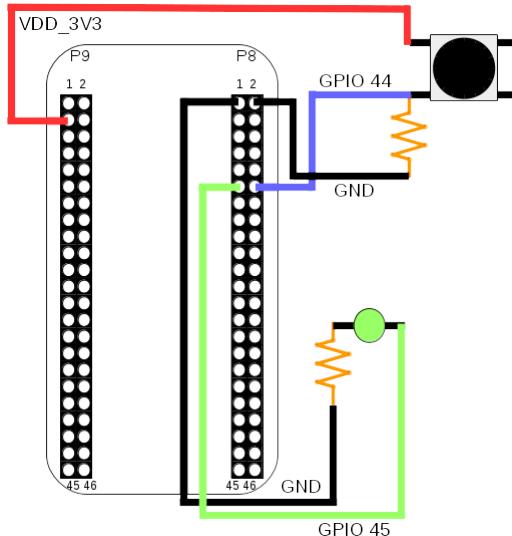


Figura 3.4: Schema per il collegamento del bottone e del led

Al fine di testare alcuni degli elementi software sviluppati, sono stati aggiunti un tasto ed un led, rispettivamente come esempio di controllo utente ed attuatore. Le evaluation board sono in genere molto pratiche per fare questo tipo di prove; l'elevato numero di porte riconfigurabili, consente di lavorare con poco sforzo con i GPIO disponibili.

Il bottone è collegato al GPIO44 (Pin 12, Header J8), utilizza come sorgente di ingresso l'uscita a 3.3V della board (Pin 3, Header J9) e portato a massa (Pin 2, Header J8) attraverso su una resistenza da 820 Ohm. Il Led verde è collegato al GPIO 45 (Pin 11, Header J8) e portato a massa (Pin 1, Header J8) attraverso una resistenza da 560 Ohm. L'uso delle resistenze a massa permette di preservare le porte, evitando di utilizzarle sempre al massimo del potenziale.

Capitolo 4

Il sistema operativo

4.1 Introduzione al sistema operativo

Il sistema operativo è un elemento software atto a gestire le risorse hardware del sistema e offrire servizi agli altri software. Frequentemente il termine “sistema operativo” è abbreviato con il suo acronimo inglese OS (Operating System). Il termine può indicare un’enorme varietà di software differenti: da sistemi semplici composti da qualche centinaia di righe di codice, a sistemi assai complessi in cui si può tranquillamente parlare di decine di milioni di linee. A causa di un uso talvolta improprio del termine, in particolare legato a produttori di alcuni sistemi operativi, nel gergo comune si tende a fare una certa confusione tra quello che è realmente il sistema operativo ed il software applicativo. In letteratura si tende comunque ad attribuire due significati alla dicitura OS: la prima prevede solo la parte del “nucleo” (o kernel), la seconda comprende alcuni software aggiuntivi quali binari, librerie, file di configurazione e script essenziali per l’avvio del sistema.

4.1.1 Classificazione degli OS

È necessario premettere che, data la grande varietà di OS esistenti e il fatto che siano progettati, sviluppati e rilasciati nei modi più svariati, non è sempre possibile applicare dei parametri di classificazione tassonomica con assoluta precisione. Tuttavia è possibile stabilire dei criteri che consentono di identificare la tipologia di OS analizzato.

Licenza

Il sistema operativo, così come la stragrande maggioranza del software comunemente rilasciato, è soggetto a rilascio secondo licenza. Una licenza è una accordo (esplicito o implicito) tra coloro che scrivono e rilasciano il software ed i destinari designati. La quantità di licenze software e loro varianti è considerevole, tale da richiedere una trattazione a se stante. Tale approfondimento però è di scarso interesse ai fini del progetto. In tal senso sarà preferita una suddivisione per “macro-categorie”, con particolare riferimento agli OS.

Licenza libera : si tratta di quei sistemi operativi in cui di solito l’uso privato è gratuito (a differenza di quello commerciale che può essere o meno

a pagamento) e di cui sono disponibili i codici sorgente. Talvolta, ma non sempre, è anche possibile sviluppare e rilasciare le proprie copie del prodotto sviluppato. Un esempio tipico di questo tipo di OS sono i sistemi GNU/Linux, rilasciati sotto licenza GPL

Licenza proprietaria : si tratta di quei sistemi per cui ogni istanza del software deve essere stata rilasciata secondo i dettami di una EULA (End User License Agreement), solitamente legate a un corrispettivo economico. Non è possibile distribuire, sviluppare o redistribuire tali sistemi operativi se non attraverso particolari accordi con il produttore. Un esempio tipico sono i sistemi Microsoft Windows.

Licenza ibrida : sono quei sistemi che sono proprietari, ma che hanno alcune componenti che sono rilasciate con licenza libera o quei sistemi liberi con componenti di rilevo rilasciati con licenza proprietaria.

Struttura del kernel

Nel corso dello sviluppo degli OS, vi è stata una continua evoluzione per quanto concerne la struttura (o architettura, anche se tale termine può generare confusione) del kernel.

Monolitici : un kernel monolitico è caratterizzato dalla presenza all'interno del nucleo, oltre che dei driver e delle strutture di multiplexing dell'hardware, anche di una serie di servizi per la gestione dei processi e della comunicazione. Tale approccio è stato utilizzato a partire dai primi kernel BSD, ed è tuttora utilizzato nei kernel Linux e FreeBSD. Se pure sia universalmente riconosciuto che non sia in assoluto il modello più efficiente, trova ad oggi grande riscontro per la facilità di sviluppo (che si traduce in costi di impiego complessivamente più bassi).

Microkernel : in un microkernel si tende a portare fuori dall'area del nucleo (e quindi fuori dalla memoria con accesso supervisore) una serie di servizi all'interno di cosiddetti "server". A loro volta i server comunicano con il kernel minimale. Nel dettaglio, si utilizzano dei meccanismi di IPC (Inter Process Communication) per lo scambio di informazioni tra i server ed il kernel. A fronte di una grande efficienza e sicurezza, si contrappone una notevole complessità di sviluppo ed una certa rigidità comportamentale complessiva. Un esempio classico di tale approccio è Minix, o parlando di sistemi embedded molto utilizzati in ambito MCU, FreeRTOS e QNX.

Ibridi : sono quei kernel che pur avvalendosi di un'architettura simile a quella microkernel, per motivi di semplicità o di miglioramento delle prestazioni, integrano all'interno del kernel minimale parti considerate "non essenziali" nella classica struttura "microkernel". Questo modello ha trovato grande riscontro in sistemi desktop/workstation commerciali quali XNU (Kernel di MacOS) e WindowsNT di Microsoft.

ExoKernel : si tratta per lo più di un nuovo approccio accademico, volto a ridurre il kernel al solo multiplexing delle risorse hardware. In pratica tutte le operazioni del sistema operativo vengono poste all'esterno del kernel in una o più libOS (librerie con compiti di sistema operativo). Tale

approccio, almeno in teoria, permetterebbe la coesistenza di più libOS (e di conseguenza di più sistemi operativi) al di sopra del medesimo Exokernel.

Per quanto alcuni microkernel abbiano trovato ampio mercato all'interno dei sistemi embedded, sono in genere utilizzati su CPU di piccole/medie dimensioni (es. le MCU) e per applicativi RealTime. All'aumentare delle dimensioni e della complessità della CPU (come avviene nei SoC) è di gran lunga preferibile utilizzare un kernel monolitico (tempi di sviluppo) e libero (dovendo rispondere a forti necessità di personalizzazione e di contenimento dei costi). Per questi motivi, per buona parte dei SoC di fascia alta presenti sul mercato, siano essi ARM o MIPS, Linux risulta essere il sistema operativo di riferimento. Nel settore mobile, per SoC dotati di GPU e/o dispositivi di Encoding/Decoding video si predilige Android rispetto a Linux. Tale scelta, più che per ragioni tecniche, è legata a fattori di collocamento sul mercato e di maggiore standardizzazione relativa alla gestione dei flussi video e delle applicazioni 3D.

4.2 I sistemi GNU/Linux

4.2.1 Storia dei sistemi GNU/Linux

Nel 1983, Richard Stallman, attivo sviluppatore presso i laboratori del MIT, lanciò il progetto GNU (G.nu is N.ot U.nix): l'intento di questo progetto era di realizzare un sistema operativo simile a Unix, ma rilasciato con licenza libera. A seguito del progetto GNU, fu fondata la Free Software Foundation, una fondazione nata per portare avanti le finalità del progetto GNU. Nonostante i risultati incoraggianti dal punto di vista di molti componenti software (uno su tutti il compilatore C, ad oggi uno dei più utilizzati in assoluto), il cosiddetto “sistema operativo GNU” risultava essere ancora privo di un kernel completo. Il kernel in sviluppo era quello che prenderà il nome di Hurd, un microkernel basato sul core Mach, ritenuto interessante dal punto di vista architettonico, ma mai giunto a maturità.

Nel 1992 Linus Torvalds, uno studente dell'università di Helsinki rilasciò con licenza libera il suo kernel (inizialmente nominato FreeX, in seguito ribattezzato Linux), ideato per essere eseguito su sistemi Minix¹. In breve periodo il kernel fu adattato per poter coesistere con l'ambiente userspace del progetto GNU, dando vita ai sistemi GNU/Linux. Ad oggi, per semplicità, si tende ad indicare con il solo termine Linux tutto il sistema operativo completo.

La licenza libera del kernel e dell'intero progetto GNU, hanno reso possibile a tantissimi sviluppatori di poter realizzare delle proprie versioni personalizzate dei singoli software o dell'intero sistema operativo, dando vita al fenomeno delle distribuzioni GNU/Linux.

Le “distribuzioni” (gergalmente definite anche “distro”) sono dei sistemi operativi completi, realizzati partendo dai medesimi codici sorgente, ma con differenze dovute a personalizzazioni e ottimizzazioni.

È possibile definire tre categorie distinte di distribuzioni:

¹Il sistema operativo basato su Microkernel scritto dal professor Tanenbaum dell'università di Vrije.

Hobbyistiche : team di sviluppo limitato a poche persone (o una sola), create e mantenute per soddisfare esigenze particolari, personalizzazioni dettagliate o per scopi didattici.

Community : realizzate attraverso il lavoro di sviluppo collettivo di “comunità” di volontari di medie/grandi dimensioni.

Enterprise : sviluppate principalmente all'interno di aziende da professionisti dedicati per sopperire alle richieste di sviluppo, mantenimento ed assistenza d'impresa, istituti finanziari ed enti pubblici.

In alcuni casi le realtà community sono sovvenzionate dal punto di vista economico dalle medesime aziende che producono prodotti enterprise; tale comportamento, apparentemente incompatibile con i modelli di business tradizionali, crea un flusso bidirezionale di idee e sviluppo software fra queste due realtà, migliorandosi vicendevolmente.

In un primo momento si poteva avere la percezione che questa tipologia di sistemi avesse un ristretto pubblico d'elezione ma, già a partire della fine degli anni '90, i sistemi GNU/Linux hanno ricevuto l'attenzione delle grandi compagnie del software e dell'hardware, arrivando ad avere una predominanza nel settore “Enterprise” (server/workstation) ed in quello “embedded” (mobile, automotive, industrial, home automation), ma non in quello desktop, dove è rimasto sempre ai margini del mercato.

4.2.2 Il kernel Linux

Linux è un kernel monolitico rilasciato con licenza libera (GPLv2). Il kernel, creato inizialmente per un'architettura x86 (per un Intel i386 a 32 bit) ad oggi supporta numerose architetture hardware differenti (x86, x86_64, IA64, arm, aarch64, alpha, mips, s390, PowerPC) nelle loro varianti a 32 o a 64 bit.

Linux è scritto in prevalenza con linguaggio C, anche se a tutt'oggi sono presenti ancora molte parti scritte in linguaggio assembly² (Sintassi GAS, AT&T). Per molti anni l'unico compilatore utilizzabile per il kernel Linux è stato GCC (anche grazie a delle opzioni speciali aggiunte appositamente nel compilatore per il kernel), ma negli ultimi anni sia Intel che Clang (LLVM) stanno sviluppando in questa direzione.

Alcune caratteristiche del kernel linux:

Preemptive Multitasking: attraverso un meccanismo basato su interrupt, viene sospeso il processo corrente, invocato lo scheduler e sulla base di un diverso livello di priorità, viene eseguito un determinato processo. In questo modo si cerca di garantire a tutti i processi un accesso pesato alle risorse hardware.

Memoria Virtuale: i processi utilizzano un spazio di indirizzamento virtuale che viene “mappato” su quello fisico, attraverso un meccanismo di traduzione indirizzi hardware (memory management unit o MMU). Semplifica le gestioni della memoria da parte delle applicazioni, migliora sicurezza e isolamento. Pur essendo possibile fare il mapping con aree di memoria non

²Linguaggio a basso livello molto vicino al linguaggio macchina (talvolta con un rapporto 1 ad 1 tra istruzione assembly e istruzione macchina).

presenti sulla RAM fisica proprio grazie al meccanismo di VM, il termine “memoria virtuale” è stato per anni erroneamente scambiato per la tecnica di swapping della memoria su disco fisso.

Loadable Kernel Module: trattandosi di un kernel monolitico è lecito aspettarsi che tutta la parte dei driver si trovi nello stesso spazio di memoria (il cosiddetto kernel space, o Ring 0). Se in passato era necessario eseguire il caricamento di tutto il kernel durante la fase di boot, con l'avvento dei moduli kernel, ovvero dei file oggetto compilati con il medesimo compilatore e headers del kernel, è possibile aggiungere in un momento secondario all'avvio il supporto per alcune periferiche. Tale strumento ha permesso di ridurre le dimensioni dei kernel caricati in fase di boot, e di selezionare semplicemente i driver necessari una volta avviato il sistema.

Symmetric Multiprocessing (SMP): la sigla SMP definisce un sistema multiprocessore omogeneo (ovvero dotato di più processori identici) in grado di condividere la stessa system-ram e i medesimi I/O, coordinati dalla stessa istanza del sistema operativo. Discorso analogo vale per i recenti sistemi multicore, che vengono trattati come singoli processori distinti.

Copy on Write: è assai frequente che più processi lavorino sugli stessi dati. Con il metodo COW (Copy on Write) si copia una sola volta in memoria per tutti i processi che usano i medesimi dati, creandone delle copie solo quando esiste il bisogno di modificarli.

Kernel API

Il kernel, per potersi interfacciare con il resto del sistema, mette a disposizione una serie di API (Application Program Interface), suddivise in due tipi distinti: le **In-kernel** api e le **kernel-to-userspace** api.

Le In-kernel sono quelle utilizzate dai cosiddetti sottosistemi: attraverso queste APIs è possibile standardizzare il comportamento e la scrittura dei driver per determinate periferiche. Importante far notare che tale standardizzazione non vale al proseguire delle versioni di kernel: un driver scritto per la versione 3.X del kernel potrebbe essere non compatibile con la versione 3.X+1 e nemmeno retrocompatibile con la versione 3.X-1.

Alcuni esempi delle In-kernel Api

Bluez: comunicazioni bluetooth

Mac80211 : per le interfacce wireless

Direct Rendering Manager (DRM): acceleratori grafici

Kernel Mode Setting (KMS): display Controller

Video4Linux (V4L): sottosistema di cattura video per Linux

Advanced Linux Sound Architecture (ALSA): sottosistema per le schede audio

Le api kernel-to-userspace invece rappresentano l'interfaccia del kernel per le cosiddette “syscall” (o chiamate di sistema). In particolare definiscono come

alcune librerie interagiscano con il kernel, cercando di offrire per quanto possibile gli standard POSIX (Portable Operating System Interface) e Single Unix System Specification. L'elenco delle syscall è assai vasto e non rientra nell'interesse specifico per la trattazione qui svolta.

Il device-tree

Il “device tree” è una struttura dati atta a descrivere particolari configurazioni hardware, utilizzata in alcuni dispositivi embedded di recente sviluppo, dotati di kernel linux versione 3.5 o successiva. In sostanza un device-tree sotto forma di sorgente può essere visto come una descrizione dettagliata di come l'hardware presente sia configurato, indicando mappature tra device e indirizzi di memoria, gestione dei pin di uscita, configurando dei “clock” o cambiando valore a particolari registri.

Come già accennato nella descrizione dell'hardware, i dispositivi embedded odierni sono in grado di trasformarsi e utilizzare non simultaneamente dispositivi diversi sui medesimi Pin a velocità differenti. Prima dell'avvento del device-tree, tali capacità erano però limitate al fatto che, ogni qual volta si dovesse configurare differentemente qualcuno dei parametri sopracitati, fosse necessaria una riscrittura di un driver e una successiva ricompilazione.

Affidando queste parametriche al device-tree, tali operazioni di riconfigurazione possono essere fatte al netto di un riavvio di sistema. In taluni casi è possibile vedere come, in base al riconoscimento di alcuni specifici segnali, un sistema possa caricare un device-tree differente, cambiando di fatto il comportamento complessivo del sistema. Il risultato finale è una maggiore libertà di sviluppo, senza la necessità di dover scendere nei dettagli dell'implementazione del codice.

Il device-tree è presente nel sorgente del kernel come file di testo, ma viene compilato sotto forma di file binario per essere caricato dal bootloader nelle prime fasi del boot. Nei primi sistemi con device-tree, i boot loader non disponevano ancora del supporto per il device-tree: in questi casi la versione binaria del device-tree veniva aggiunta in fondo al file immagine compresso del kernel³.

4.3 Root Filesystem

Come è facilmente intuibile, il sistema non è composto esclusivamente dal kernel che, per quanto indispensabile, da solo non è sufficiente. Altri elementi vanno a comporre quello che è l'insieme del software essenziale alla definizione del sistema: la libc (la libreria C) e l'interprete di comandi (la shell nella maggior parte dei casi) e alcuni tool essenziali. Questi elementi sono collocati all'interno del cosiddetto “root filesystem” (o rootfs). Contrariamente a quanto il nome possa suggerire, con questo termine non si specifica nessuna tipologia di filesystem, ma il contenuto. Con “rootfs” infatti si indica la partizione (o sistema di storage locale/remoto) all'interno della quale è possibile identificare gli elementi necessari all'avvio del sistema. Tale partizione viene solitamente montata come primo nodo dell'alberatura delle cartelle, che prende il nome di root (dall'inglese radice, indicata nel sistema con la “”). Occorre precisare che si tratta di un

³Per quanto tale metodo sia tutt'ora utilizzabile è considerato obsoleto e poco raccomandabile.

collocamento assolutamente convenzionale e che, con le opportune modifiche, si potrebbe avere un sistema le cui componenti siano altrove rispetto alla posizione della partizione “”.

4.3.1 La libreria C

Una libreria C è un elemento fondamentale all’interno di un sistema basato su kernel Linux, e rappresenta l’interfacciamento primario tra le kernel-to-userspace APIs e il software userspace. Nella maggioranza dei casi, un sistema GNU/Linux privato della libreria C, non è in grado di funzionare correttamente. Esistono più tipi di libc, alcune più generiche, altre dedicate a dei settori specifici.

GNU C Library (glibc): in assoluto la libreria C di riferimento nei sistemi GNU/Linux, la più completa e di conseguenza anche la più ingombrante.

eglibc: variante della nota glibc pensata specificatamente per ambienti embedded. Lo sviluppo è fermo al 2014.

musl: disegnata per essere particolarmente pulita ed efficiente, è stata progettata per preservare la possibilità di una compilazione statica (ambienti realtime). Compatibile con lo standard Posix 2008 e C11, è utilizzata in sistemi linux pensati per i router (ex. openWRT).

uClibc: libreria C pensata appositamente per essere estremamente compatta (la u sta per μ , ad indicare “micro”). Inizialmente era stata progettata per essere la libreria C di μ -Linux, la versione ridotta pensata per sistemi non dotati di MMU.

Bionic: è la versione libc ideata da Google per i sistemi operativi basati su Android; dalle dimensioni ridotte e dal codice ottimizzato, al momento non risulta essere compatibile con gli attuali standard POSIX.

4.3.2 Init

Nei sistemi operativi di derivazione diretta ed indiretta di Unix, “init” rappresenta il primo processo del sistema operativo (il suo PID⁴ è convenzionalmente uguale 1). Nel modello gerarchico “padre/figlio” dei processi, init risulta essere il processo “antenato” di tutti i processi in esecuzione. Tale processo rimane attivo dall’avvio del sistema, sino alle fasi di riavvio o spegnimento. Il tentativo di chiudere questo processo, porta ad un errore denominato “kernel panic”⁵.

Un sistema di init è invece l’insieme di strumenti software e meccanismi di configurazione che comprende anche il processo init: tale sistema consente l’avvio di tutti gli applicativi e demoni previsti dal sistema. In alcuni casi limite è possibile non utilizzare un vero e proprio processo di init, usando il’alternativa una shell minimale.

Tra le caratteristiche comuni ai sistemi di init, esiste il concetto di runlevel, ovvero l’identificazione di diverse modalità operative con cui chiamare il processo: tali runlevel, solitamente indicati con un numero progressivo da 0 a 6,

⁴Process Identification Number.

⁵Un “kernel panic” è un errore irreversibile nei sistemi operativi Unix-like che comporta lo stop forzato di ogni processo di sistema e del kernel.

possono assumere un significato differente sulla base della configurazione, anche se come standard de-facto (non sempre rispettato alla lettera) si identificano nel seguente modo.

0 (Halt): è il runlevel chiamato per eseguire lo spegnimento del sistema

1 (Single): il runlevel chiamato per avviare il sistema in “Single User Mode”, ovvero caricando il numero minimo di servizi indispensabili per offrire un’interfaccia testuale non multiuser.

2: non specificato. In alcuni sistemi è la modalità multiuser senza il supporto di rete.

3 (Multiuser): in genere il sistema avviato nella sua interezza, multiutente, servizi di rete e demoni, ma senza interfaccia grafica.

4: non specificato

5 (GUI): equivalente alla modalità multiuser, con l’aggiunta dell’interfaccia grafica.

6 (Reboot): il runlevel chiamato per eseguire il riavvio del sistema.

In alcuni sistemi di init, pur non esistendo un vero e proprio supporto al meccanismo dei runlevel, questo viene implementato attraverso degli script atti ad emularne il comportamento.

Nel corso degli anni vi sono state numerose variazioni ai sistemi di init utilizzati dalle distribuzioni GNU/Linux; per quanto possa sembrare una scelta ininfluente sull’utente finale, la scelta del sistema di init ha rappresentato un fattore di caratterizzazione delle distribuzioni stesse. Anche nel caso del progetto oggetto di questa trattazione, non si può prescindere dalla scelta del sistema di init: al fine di comprenderne le differenze verranno presentati alcuni esempi di init.

BSD Init: di derivazione dai sistemi BSD Unix è basato su una serie di script presenti nella cartella “/etc” (rc). Il processo init esegue sequenzialmente le operazioni descritte all’interno dei vari script. Inizialmente non disponeva del supporto per i runlevel, aggiunto nelle versioni più recenti.

Sys-V init: di derivazione dai sistemi Unix System-V da cui prende il nome, ha introdotto il sistema dei runlevel e, nei sistemi moderni, dispone di una cartella specifica per ogni runlevel all’interno della cartella “/etc”. Anche in questo caso si tratta di esecuzione sequenziale.

systemd: systemd è un complesso sistema di librerie, tool e file di configurazione che, tra le altre cose comprende anche il sistema di init. Di recente concezione, e scelta predefinita per le maggiori distribuzioni Linux desktop/server, offre meccanismi di caricamento dei servizi in parallelo e la gestione di dipendenza ed ottimizzazioni specifiche per le architetture multiutente. Il suo utilizzo nel settore embedded è ancora in discussione, anche se la sua maggiore complessità nella gestione e l’inefficacia di alcune ottimizzazioni lo renderebbero poco adatto ai sistemi di piccole dimensioni.

Busybox Init: integrato all'interno del binario di busybox, offre un meccanismo semplice, basato su script, per l'avvio dei servizi e del software. Ideale per piccoli sistemi senza grandi modifiche a tempo di esecuzione, mostra forti limiti di gestione su sistemi di dimensioni maggiori.

4.3.3 La shell

Parlando d'interazione con l'utente si possono distinguere due differenti tipologie di interfacce: le cosiddette CLI (Command Line Interface) strettamente testuali o quelle di tipo grafico (Graphical User Interface o GUI). Nella pratica comune con il termine shell si tende a indicare la prima tipologia, ovvero quella testuale. È necessario sottolineare l'opportuna distinzione tra shell e terminale, usati erroneamente come sinonimi. Nel primo caso si tratta di uno software attivo all'interpretazione dei comandi, parlando di terminale si fa invece riferimento ad un dispositivo hardware (della tipologia dei cosiddetti dispositivi a caratteri, come ad esempio la tastiera o le porte seriali) di comunicazione, sia esso reale o emulato. Fatta questa opportuna distinzione, rimane comunque valida la dicitura "comandi da terminale" per indicare l'uso di comandi all'interno di una CLI.

A partire dai primi sistemi UNIX, gli OS sono stati sempre dotati di una (o più) shell per interfacciarsi con l'utente e permettere di eseguire task in modo interattivo (lanciando i comandi manualmente) e non interattivo (ponendo i comandi in dei file di testo per un'esecuzione automatica). In molti casi le shell definiscono un vero e proprio linguaggio (definito scripting) che possa essere interpretato durante l'esecuzione. I file contenenti i comandi da interpretare sono definiti script.

In ambienti Unix/Linux possono essere presenti una o più delle seguenti shell:

Bourne Shell: scritta da Stephen Bourne e rilasciata nel 1977, apparve per la prima volta all'interno di Unix versione 7, era indicata generalmente con il nome di "sh" (il nome del file binario per richiamarla).

C-Shell: rilasciata alla fine degli anni '70 per i sistemi BSD (Berkeley Software Distribution), C-shell deve il suo nome allo sforzo compiuto dagli sviluppatori per far sì che il suo linguaggio di scripting fosse il più possibile aderente alla sintassi del più noto linguaggio di programmazione C, considerato più leggibile[28].

Bourne again shell: nata nel 1989 come sostituzione della Bourne Shell (da cui prende il nome), ha trovato grande applicazione in buona parte dei sistemi Unix/Linux esistenti. Ad oggi è considerata la shell predefinita dei sistemi GNU/Linux, ma non in ambito embedded, dove risulta sovradimensionata. Viene indicata con il nome bash.

ash: nata alla fine degli anni '80 per sostituire la Bourne Shell e sviluppata da Kenneth Almquist (da cui prende il prefisso 'a'). Nonostante non sia completa come bash, è stata particolarmente apprezzata nei vari ambienti embedded per la sua compattezza, tanto da essere la scelta predefinita nei sistemi basati su busybox.

Korn Shell: basata su Bourne Shell, molto completa ed estensibile, ha trovato applicazioni in ambienti server, in particolare per il suo linguaggio di scripting, che ben si sposa con ambienti DB.

4.4 Bootloader

Un bootloader è un software creato per consentire il caricamento del kernel, la selezione dei parametri di avvio e l'avvio del sistema. Per adempiere a tale scopo è necessario che questi siano in esecuzione prima del sistema operativo stesso. In sistemi di elaborazione di complessità superiore alle MCU, o comunque che eseguano sistemi operativi completi, la presenza di un bootloader è da considerarsi necessaria. Nei personal computer, così come nei sistemi workstation e server, è assai comune che ogni sistema operativo abbia un proprio bootloader. All'occorenza e se opportunamente configurato, un bootloader può prestarsi anche all'avvio di sistemi operativi differenti da quello "predefinito". Nel mondo GNU/Linux sono famosi Grub e Lilo, mentre i sistemi Apple e Microsoft utilizzano un loro software proprietario.

Un bootloader è a sua volta un piccolo sistema operativo, in quanto deve disporre degli elementi software necessari (potremmo definirli impropriamente dei driver) per poter accedere ai dispositivi di storage che contengono il sistema operativo (o parti di esso) ed in alcuni casi anche dispositivi di rete, porte seriali ed altre porte di comunicazione.

Nella maggioranza dei casi, nei PC (o in elaboratori accostabili a tale categoria), si dispone di un BIOS (ovvero di un firmware residente in una memoria FLASH/EEPROM della scheda madre) atto ad inizializzare l'hardware nel modo corretto. Tra i compiti di un BIOS⁶ c'è la configurazione dei clock di sistema (velocità CPU, RAM, BUS), il rilevamento dei dispositivi di storage, l'inizializzazione di dispositivi di comunicazione, schdede grafiche e gestione energetica. In sostanza, un bootloader per un computer tradizionale si trova parte del lavoro già svolto, limitandosi a fornire il corretto interfacciamento per l'utente per consentire la scelta del sistema e il passaggio dei parametri. Nei sistemi di ultima generazione i classici BIOS sono stati sostituiti dai più moderni UEFI⁷, ancora più completi e complessi, tali da richiedere dello spazio sul sistema di storage principale.

Nei sistemi embedded la situazione è decisamente più complicata, in quanto, ad esclusione di casi eccezionali, il più delle volte non dispongono di BIOS, bensì di assai più limitati e compatti BootROM.

Un bootROM esegue il numero minimo di operazioni per poteter "puntare"⁸ verso una specifica area di storage. Questo vuol dire che il bootloader deve farsi carico di tutte le operazioni necessarie per la prima configurazione del hardware al fine di poter caricare il sistema operativo. Tra le altre cose c'è da considerare anche il fatto che per determinate esigenze ingegneristiche, questi dispositivi devono essere in grado di poter avviare il sistema utilizzando sorgenti assai diversificate tra di loro quali memorie NAND/NOR/FLASH, SDCard, USB

⁶BASIC I.nput O.utput S.ystem

⁷U.nified E.xtensible F.irmware I.nterface.

⁸Termino gerale derivato dalla programmazione imperativa, sta ad indicare l'operazione di impostare il program counter della CPU su un indirizzo di memoria.

(host, device ed OTG), SATA (assai raramente), ethernet o addirittura delle connessioni lente quali serali UART ed SPI.

La complessità di tale compito ha portato molti produttori a suddividere il bootloader in due componenti separati, definiti a loro volta pre-bootloader e bootloader: il primo si occupa delle inizializzazioni basilari quali la CPU, la RAM, i BUS ed un primo dispositivo di storage per “puntare” al bootloader completo. Quest’ultimo, una volta caricato, oltre ad avere un maggior supporto per i vari dispositivi presenti, offre anche una CLI⁹, che può essere interattiva o meno, per poter selezionare il dispositivo e i parametri di boot.

I bootloader, per evidenti restrizioni dal punto di vista della dimensione del codice, non possono essere flessibili allo stesso modo di un kernel. Ciò comporta la necessità di avere un bootloader specifico per ogni differente dispositivo, ottimizzato e specializzato nella configurazione hardware presente.

4.4.1 U-boot

U-Boot (nome completo Das U-boot, testualmente “Il bootloader universale”) è un progetto rilasciato con licenza libera GNU GPLv2. Solitamente classificato nella categoria “firmware”, è uno dei bootloader più utilizzati nel mondo dei SoC. Esistono numerosi porting di U-boot per un ampio insieme di architetture e SoC differenti. La licenza aperta e la notevole semplicità dal punto di vista della programmazione ne hanno consentito l’integrazione in numerosi progetti, rendendolo “de-facto” il termine di riferimento nel suo settore.

L’estrema possibilità e personalizzazione hanno reso questo software molto suscettibile, dal punto di vista dell’utilizzo, alle modifiche apportate dai produttori dei vari SoC. Nonostante sia possibile riconoscere alcuni elementi comuni, è assai probabile che una serie di comandi funzionati sull’istanza del bootloader presente su una eval board di Texas Instruments, non sia eseguita correttamente su una board Freescale o ST.

In alcune implementazioni, un sottoinsieme degli elementi di U-boot è stato utilizzato per costruire il pre-bootloader; sfuggitandone in parte il codice a basso livello, il pre-bootloader viene compilato internamente al sorgente di U-boot (es Xloader di ST o MLO di Texas Instruments).

In base a quanto abilitato dal produttore del singolo SoC, u-boot consente di caricare un’immagine di sistema e del device-tree da diverse fonti.

All’interno di U-boot, se abilitata, esiste un ambiente CLI per l’esecuzione di comandi, script e per le impostazioni di variabili di ambiente per gestire la fase di boot. U-Boot mette a disposizione un tool userspace per la creazione di file di immagine kernel (mkimage) conformi allo “U-Boot image format”.

U-boot Image Format

Lo U-boot Image format è un insieme di dati, solitamente contenuti nei primi 72byte del file immagine, contenente alcuni informazioni utili a u-boot per eseguire la fase di caricamento.

A seguire una rapida spiegazione dei parametri configurati nello “u-boot image format”[36].

⁹Command Line Interface.

Target Operating System: specifica il sistema operativo utilizzato (es Linux, NetBSD, VxWorks, QNX, RTEMS, ARTOS, Unity OS, Integrity)

Target CPU Architecture: specifica l'architettura (es. ARM, AVR32, BlackFin, M68K, Microblaze, MIPS, MIPS64, NIOS, NIOS2, Power Architecture®, SuperH, Sparc, Sparc 64 Bit, Intel x86)

Compression type: specifica la compressione (uncompressed, gzip, bzip2, lzo)

Entry point: il punto in cui “salterà” il bootloader per avviare il kernel.

Image name: campo opzionale.

Image Timestamp: data e ora relative al momento della creazione dell'immagine.

4.5 La fasi di boot

Quanto visto sin ora ci permette di valutare come e quando gli elementi presentati sin ora (pre/Bootloader, Kernel, Rootfs, Init) interagiscano tra di loro per arrivare a un sistema funzionante.

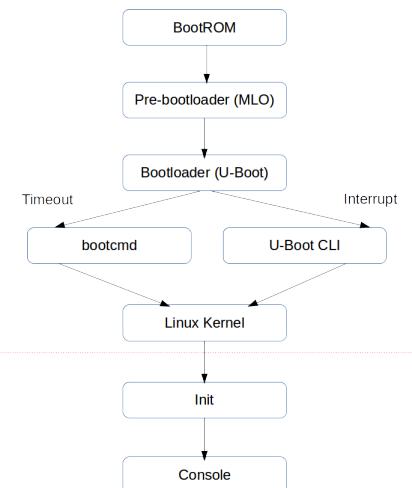


Figura 4.1: Schema basilare delle fasi di boot

Si può considerare la fase di boot come il susseguirsi di una serie di software (anche se per alcuni di essi potrebbe essere più attinente la definizione di firmware), ognuno con determinati compiti specifici (ma non unici), al fine di arrivare ad un sistema funzionante. Dopo le prime tre fasi, quelle inerenti a BootROM, pre-bootloader e bootloader (vedere sezione 4.4), avviene il caricamento del kernel. Il bootloader provvede al passaggio di alcuni parametri di configurazione da parte del bootloader (in alcuni sistemi tali parametri sono inclusi all'interno della configurazione del kernel). Il kernel, eseguite le operazioni d'inizializzazione e configurazione, “monta”¹⁰ il rootfs, collocato solitamente nella posizione radice “/”. Il passo successivo è quello di cercare il binario di “init”

¹⁰Termine appartenente al gergo informatico, indica l'azione di mappatura un filesystem all'interno dell'alberatura principale del rootfs.

ed eseguirlo. Da questo momento in poi il controllo rimane ad init che, come già visto in precedenza, effettuerà l'inizializzazione del sistema userspace sino al raggiungimento di una shell o di un'interfaccia grafica. Si tratta per lo più di uno scenario tipico, in quanto, in casi estremi, si saltà il processo di init per arrivare direttamente ad una shell (tramite parametri di boot). In altri casi, in sistemi progettati per non avere alcun interprete di comandi attivi per motivi di sicurezza, il processo di init porta direttamente all'esecuzione di un particolare software selezionato.

Alexjan Carraturo

Capitolo 5

Implementazione

In questa sezione verranno spiegati i passaggi per l'implementazione del dispositivo embedded e del suo sistema.

5.1 Implementazione Hardware

L'implementazione dell'hardware in quanto tale non rientra tra gli scopi principali di questa tesi, ciò nonostante vi sono alcune scelte hardware che si riflettono significativamente sulle scelte di sviluppo software, e quindi non trascurabili.

5.1.1 Form Factor

Nella progettazione di un dispositivo embedded è importante considerare le dimensioni ed il peso di un oggetto: volendo realizzare un oggetto utilizzato fattivamente da un utente è necessario assicurarsi che esso sia facilmente portatile, non sia ingombrante e non sia pesante. In tal senso va considerata la presenza di una batteria sul dispositivo, che sia capace di alimentarlo per un tempo ragionevole, ma che al contempo possa preservare per quanto possibile la portabilità ed il costo.

Stabilito il consumo del dispositivo finale (operazione non banale, raramente paragonabile a quello della evaluation board), si può scegliere la dimensione della batteria da utilizzare anche in relazione al PCB finale.

5.1.2 PCB

In una evaluation board sono presenti un gran numero di componenti non necessari per il singolo progetto. Questo permette di rimuovere questi componenti dal punto di vista hardware, liberare delle porte di I/O e realizzare un PCB più compatto ed ottimizzato, sia per quanto concerne gli spazi che per il consumo.

Questa scelta si riflette anche sulla compilazione del kernel e dei singoli tool software. Facendo l'esempio pratico del progetto, eliminando l'uscita HDMI dal sistema finale e riconfigurando il cosiddetto device-tree (vedere sezione) si liberano un gran numero di pin utilizzabili per altro. Rimuovendo l'uscita video, potrebbero risultare inutili sia i driver dell'acceleratore grafico (quindi disattivabile sul SoC, garantendo un discreto risparmio energetico) sia tutto il software relativo all'interfacciamento grafico (alleggerendo molto il peso del sistema finale).

L'organizzazione degli elementi su un PCB è argomento complesso ed articolato che, per quanto interessante, non sarà tema di questa trattazione.

5.1.3 Costo

Se dal punto di vista accademico si tende in genere a trascurare questo fattore, dal punto di vista produttivo e commerciale rappresenta un fattore vitale. Riuscire ad eliminare componenti non necessarie, o a selezionare componentistica più economica per il prodotto finale, può fare la differenza tra un prodotto in produzione ed uno rimasto sulla carta. Tutto questo però deve essere pensato e calcolato in un giusto compromesso con l'efficienza termica ed elettrica delle componenti e, cosa non secondaria, con la loro affidabilità. A partire dalla scelta del SoC infatti, si cerca il miglior compromesso tra le caratteristiche tecniche offerte ed il progetto finale, minimizzando gli sprechi. Tali considerazioni, unite a fattori di affidabilità e di revisioni del silicio spiegano come talvolta, progetti apparentemente non vincolati strettamente a ragioni di prezzo, siano sviluppati su piattaforme più datate e meno performanti. Bisogna inoltre considerare che per quanto riguarda alcuni dispositivi, oltre al costo di produzione fisico, risultano considerevoli anche le royalties (ovvero diritti economici dell'azienda sviluppatrice delle singole IP).

5.1.4 Hardware finale

Se pur non sia possibile concretamente definire la struttura finale ed il layout del PCB, è possibile, in base alle considerazioni fatte sin ora prevedere quello che sarà l'organizzazione di massima del prodotto finale.

Basandosi sui possibili scenari di utilizzo ipotizzati per il dispositivo (vedere sezioni 7.1, 7.2 e 7.3) e le successive scelte sul piano dell'architettura software, si può infatti stilare una lista di differenze con la eval board e l'hardware utilizzato, pur mantenendo una quasi completa compatibilità dal punto di vista software:

- Rimozione del fisico ethernet
- Rimozione della presa HDMI
- Rimozione della eMMC
- Rimozione della USB Host
- Collegamento di un chip Realtek8187SE SDIO/Serial per le funzioni WiFi e Bluetooth.
- Mantenimento della USB OTG, del lettore MicroSD card
- Aggiunta Antenna su PCB

Come già detto in precedenza, la rimozione degli elementi fisici e relativa disabilitazione a livello software consente di liberare alcune porte di collegamento: nello specifico, laddove era collegata la eMMC è possibile collegare la nuova scheda combo Wifi/Bluetooth (SDIO utilizza un subset dei pin utilizzati per il collegamento dei fisici delle SD card).

Con una certa approssimazione, e in considerazione della estrema variabilità dei prezzi sul mercato, si può stimare un prezzo al produttore di circa 20-25\$ al pezzo per produzioni sopra le mille unità.

5.2 Implementazione Software

In questa sezione verrà illustrato in dettaglio come riprodurre, partendo da zero (o quasi) tutto il sistema funzionante. Trattandosi di un sistema embedded molto personalizzato, è assai difficile utilizzare immagini di sistemi preesistenti: in tal senso è opportuno ricostruire il sistema nella sua quasi totale interezza. Ciò comporta che la totalità degli elementi sia ricompilata partendo dai codici sorgenti, configurati e modificati sulla base delle necessità.

5.2.1 Toolchain

Trattandosi di compilazione da codice sorgente, è necessario disporre di un compilatore efficiente e completo. Una toolchain è solitamente composta dai seguenti elementi

Compilatore: un software che trasforma il codice sorgente (solitamente testuale) nel cosiddetto codice oggetto (comunemente binario ma non eseguibile, nel linguaggio C spesso riconoscibile dall'estensione “.o”)

Linker: questo elemento esegue il linking tra i vari codici oggetto prodotti e le librerie utilizzate dal programma, trasformandoli in codice eseguibile.

librerie: deve disporre delle librerie necessarie alla fase di linking, in particolare per quanto riguarda le librerie d’interfacciamento con il sistema operativo.

debugger: opzionale in alcune toolchain, è lo strumento atto ad eseguire il debug (individuazione dei problemi) dei software compilati.

Una toolchain inoltre deve essere “compilata” per l’architettura selezionata, in quanto, in generale, non è possibile utilizzare la medesima toolchain su architetture differenti. Nel caso dei sistemi embedded la questione si complica ulteriormente, dato l’elevato livello di differenziazione tra le varie architetture e tra i singoli SoC.

La necessità di avere una toolchain specifica per l’architettura obiettivo (comunemente definita con il termine “target”), porta ad un’ulteriore distinzione:

Toolchain native: una toolchain nativa è a sua volta stata compilata ed eseguita in un sistema con la medesima architettura hardware/software; se nel campo dei sistemi desktop workstation questa è la scelta predefinita, risulta abbastanza nuova e controversa nei sistemi embedded. Da un lato offre una maggiore semplicità per i sistemi di compilazione (a partire dai cosiddetti autotools), al contempo però richiede una notevole potenza computazionale non sempre propria dei sistemi embedded, oltre che di un sistema operativo già funzionante sull’architettura selezionata, requisito non sempre soddisfacibile. Al momento un simile approccio è utilizzato, oltre che per i sistemi Desktop, Workstation e Server, solo per il mantenimento di grandi distribuzioni GNU/Linux per SoC di fascia molto alta, non infrequentemente utilizzando batterie di SoC dedicati solo alla compilazione.

Cross-Toolchain: una cross-toolchain è costituita dagli stessi elementi di una toolchain nativa, compilati per una architettura ospite (detta anche “host”),

ma configurata per produrre binari per una architettura differente. Tale scelta è assai più comune in ambito embedded, in quanto, pur richiedendo un maggior lavoro di configurazione, permette di sfruttare sistemi dotati di maggiore potenza per produrre i binari. Inoltre, non ha come prerequisito un sistema già funzionante per l'architettura target. Per alcune architetture di fascia minore è l'unica alternativa. In molti casi, i produttori del SoC forniscono delle cross-toolchain già pronte, configurate ed ottimizzate per il sistema selezionato.

Considerati i limiti computazionali imposti, e la necessità di dover ricostruire un sistema da zero, la “cross-compilazione” risulta la scelta migliore. La compilazione di una cross-toolchain è una operazione complessa, computazionalmente costosa, e non sempre redditizia sul piano delle performance. Può risultare una scelta vantaggiosa invece, quando possibile, l'utilizzo di una cross-toolchain precompilata; oltre agli evidenti vantaggi dal punto della semplicità e del minor tempo impiegato, si tratta di toolchain già abbondantemente testate da chi sviluppa l'architettura e successivamente da una larga base di utenti. Parlando di architetture ARM a partire dalle ARMv7 in poi, le toolchain prodotte da Linaro¹ sono tra quelle più utilizzate.

5.2.2 L'ambiente di sviluppo

Tra gli scopi di questo progetto c'è, oltre che la progettazione del sistema hardware/software in grado di essere un nodo di una rete mesh, anche di fornire delle semplici procedure di sviluppo, manutenzione e modifica del sistema stesso. A tale scopo è necessario produrre una serie di strumenti facilmente utilizzabili e configurabili. Per quanto l'idea di produrre tutto interamente da zero (gergicamente “from scratch”) sia didatticamente interessante, risulta essere poco pratica dal punto di vista della portabilità e la manutenzione del progetto. Per tale scopo sono preferibili degli ambienti di compilazione automatizzati, molto utilizzati dalla stragrande maggioranza degli sviluppatori di sistemi embedded. Quelli attualmente più utilizzati sono:

Yocto: lo Yocto Project[34], promosso dalla Linux Foundation e fortemente basato su OpenEmbedded, nasce con lo scopo di semplificare la creazione di distribuzioni linux per sistemi embedded, indipendentemente dall'architettura di destinazione. Yocto si basa sul concetto di “Layer” ovvero di una serie di strati non necessariamente in gerarchia verticale, contenenti della metà informazioni sulla compilazione del sistema e dei suoi componenti. Se da un lato quindi può risultare più onerosa la scelta di yocto rispetto ad altri sistemi di build, dall'altro Yocto è quello che si presta maggiormente ad essere utilizzato con software SCM (Software Control Management), quindi adatto per quei progetti realizzati in team di sviluppo numerosi.

Buidroot: buildroot[7] è un insieme di script, Makefile a patch pensati per generare un sistema operativo Linux su piattaforme embedded. Buildroot

¹Linaro è un gruppo di lavoro che nasce dalla collaborazione di diverse aziende operanti ad alto livello nel settore embedded legate alle architetture ARM (es. ARM, HISILICON, SPREADTRUM, ST, Texas Instruments, Mediatek, ZTE.) per condividere e standardizzare quanto più possibile gli strumenti di sviluppo.

può generare la cross-toolchain, il root filesystem, le immagini del kernel e dei vari bootloader. Nell'ottica di sviluppo di buildroot ci sono da sempre i sistemi embedded più piccoli e semplici (inizialmente era utilizzato come semplificazione per gli sviluppatori che lavoravano su ambienti uKernel e ulibc, per sistemi privi di MMU) e la possibilità di offrire un valido ambiente di sviluppo manutenibile da un singolo sviluppatore. Supporta una grande varietà di architetture, quali ARC, ARM (32/64 LE/BE), Blackfin, Microblaze, Mips (32/64 BE/LE), Nios, PowerPC (32/64), SuperH, Xtensa ed ovviamente le architetture x86(32/64)².

Prerequisiti per l'ambiente di sviluppo

Esistono un considerabile numero di possibili variazioni sul tema degli ambienti di sviluppo, in questo caso verranno descritti brevemente i prerequisiti base.

- Un computer x86_64 (Cpu Core2 Duo o superiore)
- 512 Mb di RAM (raccomandati 8Gb)
- Almeno 10Gb di spazio disponibile sul dispositivo di storage
- Sistema operativo GNU/Linux recente

Il sistema operativo utilizzato per la fase di sviluppo è un Fedora GNU/Linux versione 22, 64bit. Oltre ai pacchetti di sistema base sono necessari i pacchetti di sviluppo (es gcc, glibc-devel, ncurses-devel). Alcune toolchain sono pre-compilate per sistemi 32bit, quindi in sistemi a 64bit è necessario avere una duplice installazione di questi pacchetti (32 e 64bit).

Configurazione dell'ambiente di sviluppo

A seguire una lista di passaggi da seguire per inizializzare l'ambiente di sviluppo. Questa lista di comandi, come gli script ed i passaggi successivi sono pensati e testati per l'utilizzo dell'interprete di comandi bash.

```
$ cd ~/  
$ mkdir mesh_system  
$ cd mesh_system  
$ wget http://buildroot.uclibc.org/downloads/buildroot-2015.08.1.  
     ↪ tar.gz  
$ tar -xvzf buildroot-2015.08.1.tar.gz  
$ cd buildroot-2015.08.1
```

All'interno della cartella di buildroot sono da identificare alcune cartelle che verranno utilizzate successivamente:

dl: la cartella dove vengono scaricati i sorgenti compressi.

fs: i makefile per la generazione delle singole immagini dei root filesystems

arch: file di configurazione per le differenti architetture hardware

²Con la dicitura 32/64 si intende il supporto per architetture a 32bit e a 64bit. Con la dicitura BE/LE si intende il supporto per le architetture Big Endian e Little Endian.

board: contiene delle sottocartelle dedicate al supporto ad un numero limitato di evaluation board, come script di configurazione e defconfig kernel/u-boot.

configs: contiene i defconfig per le evaluation board più diffuse, oltre che per alcune architetture specifiche

output: una volta effettuata la fase di “build”, contiene le immagini di sistema generate, i binari della toolchain, e il rootfs temporaneo.

system: i file di configurazione per la generazione del sistema.

toolchain: contrariamente a quanto potrebbe far pensare il nome, questa cartella non conterrà la toolchain, ma solo i file di configurazione per la sua generazione/download.

docs: file di documentazione

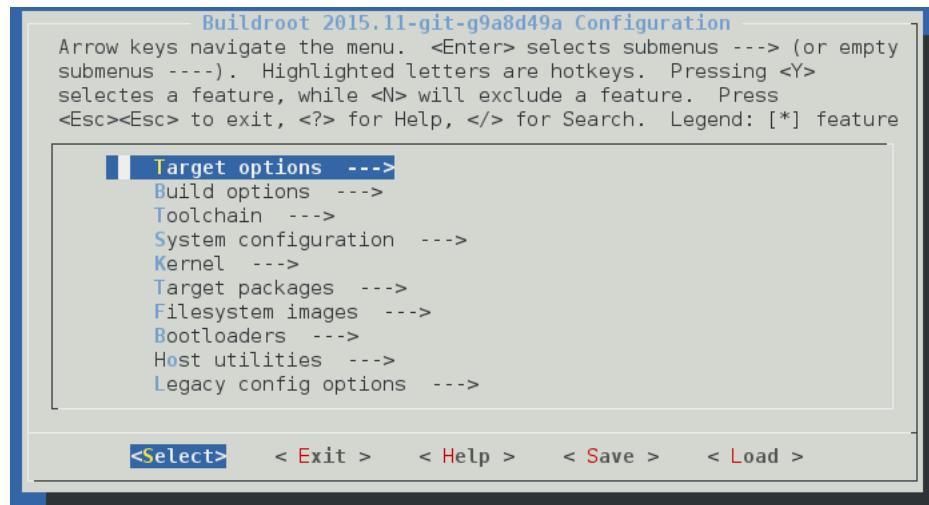


Figura 5.1: Menu principale buildroot

I 3 primi passaggi sono necessari per la creazione di una cartella di lavoro. In questa cartella verranno sviluppate tutte le singole parti del progetto, per poi essere assemblate in un unico sistema di build.

Il comando wget consente di scaricare uno specifico pacchetto da internet all'interno della cartella di lavoro. Nella specifico, la versione 2015.08.1 in formato compresso “tar.gz”.

Il comando tar utilizzato in questo modo estraе il pacchetto compresso generando la cartella dentro la quale si troverà il buildroot (buildroot-2015.05).

Una volta entrati all'interno della cartella buildroot è possibile lanciare il comando "make beaglebone_defconfig" che preconfigura l'ambiente di compilazione seguendo le specifiche scritte nel file "beaglebone_defconfig".

```
$ make beaglebone_defconfig
$ make menuconfig
```

L'uso dei file definiti "defconfig" è abbastanza comune, sia in buildroot, sia nel sorgente del kernel, da cui tale struttura è stata ispirata: Questi file contengono delle configurazioni basilari (solitamente bastevoli alla creazione di un sistema minimale, ma non sufficienti ai fini di un progetto completo) specifici per alcune evaluation board, device embedded o sistemi di emulazione (es. qemu). È giusto ricordare che in relazione all'hardware presente sul mercato, solo una piccola parte è supportata con un proprio defconfig. In molti casi, in assenza di uno specifico defconfig, è necessario ricreare un file di configurazione da zero. Inoltre, un defconfig può essere una versione generica per un insieme di dispositivi, quindi contenere elementi non utili all'hardware specifico o non perfettamente aderenti al progetto considerato. In generale è considerato fondamentale procedere ad una configurazione manuale di buildroot, eseguendo successivamente il comando "make menuconfig"³.

Osservando il menu principale di buildroot è possibile distinguere le seguenti voci

Target options: in questa sezione vengono specificate la caratteristiche dell'architettura e del formato binario.

Build options: opzioni relative alle sezioni di download, compilazione e debug del sistema prodotto

Toolchain: in questa sezione viene selezionato il tipo di toolchain, e nel caso di una toolchain costruita da zero anche le versioni della libreria C e delle cosiddette "binutils".

System Configuration: parametri di configurazione base del sistema, tra cui il sistema di init, le opzioni di boot e script di pre e post configurazione

Kernel: creazione di un'immagine del kernel, specificando la locazione del sorgente, uno specifico defconfig ed il tipo di immagine da utilizzare.

Target Packages: selezione dei pacchetti che saranno presenti all'interno del sistema finale.

Filesystem Images: selezione dei diversi modi di generare le immagini del rootfs.

Bootloaders: è possibile selezionare la compilazione di uno specifico bootloader (vedere sezione u-boot).

Host utilities: è possibile compilare alcune applicazioni locali per il sistema host. In molti casi non è necessario operare in questa sezione.

Legacy config options: in questa sezione sono presenti alcune opzioni rimosse dalla versione attuale di buildroot, ma configurabili per motivi di retro-compatibilità.

³Sia per quanto riguarda buildroot che per il sorgente del kernel linux, menuconfig è un tool binario compilato al momento dell'esecuzione del comando "make menuconfig". Per essere compilato necessita dei pacchetti ncurses (libreria per menu grafici da console testuale, derivata da curses) ed ncurses-devel (gli header file per lo sviluppo).

In questa prima fase di build è necessario un sistema minimale (quindi con le opzioni di default) e cosa più importante, il cross-compilatore da utilizzare durante tutte le fasi successive. La scelta fatta nel defconfig è quella della toolchain compilata da zero, che mal si presta a quelli che sono i bisogni dello sviluppo. Per questo motivo è necessario entrare nel menu toolchain, selezionare una toolchain esterna e selezionare la toolchain precompilata Linaro 2014.09. Per scopi di debug, potrebbe risultare utile in fase di sviluppo abilitare la voce “Copy gdb server to the Target”⁴. Tale opzione, in fase di rilascio del software definitivo dovrà essere rimossa in quanto non essenziale per il prodotto finale. Al fine di eliminare problemi successivi in fase di compilazione e testing del sistema, risulta opportuna anche la selezione dell’opzione “tar the root filesystem”, selezionando gzip come algoritmo di compressione all’interno del menu “Filesystem Images”.

Una volta salvato il file di configurazione ed usciti da “menuconfig” eseguire i comandi

```
$ make source  
$ make
```

Il comando “make source” scarica automaticamente⁵ i sorgenti che dovranno essere ricompilati. Il comando “make” avvia la compilazione del sistema, delle dipendenze e crea una prima immagine del sistema e del kernel.

Una volta ultimata la compilazione, oltre ad una versione preliminare del sistema, sarà disponibile anche la toolchain utilizzabile esternamente. La toolchain binaria non sarà disponibile nei “path” del sistema host: per questo motivo è importante creare uno script che al bisogno esporti determinate variabili di ambiente.

```
$ cd ..  
$ touch toolchain.source  
$ echo "export ARCH=arm" > toolchain.source  
$ echo "export CROSS_COMPILE=arm-linux-gnueabihf -" >> toolchain.  
    ↪ source  
$ echo "export PATH=\$PATH:\$HOME/mesh_system/buildroot-2015.08.1/  
    ↪ output/host/opt/ext-toolchain/bin" >> toolchain.source
```

5.3 La scelta e compilazione del kernel

5.3.1 Vanilla, Mainline, Longterm e SoC version

Nonostante buildroot offre un proprio servizio di compilazione del kernel, pre-configurato attraverso il defconfig, è assai frequente che tale configurazione faccia riferimento a kernel datati, testati sulla piattaforma senza alcune personalizzazioni, e probabilmente inadatto all’uso su un progetto nuovo. Nel caso di modifiche hardware (come l’aggiunta del supporto hardware per alcune schede wifi) o l’aggiunta di alcuni protocolli di rete non è possibile utilizzare la versione preconfigurata, ma è opportuno generare un nuovo defconfig per il kernel da aggiungere successivamente alla compilazione automatizzata. Buildroot permette di utilizzare defconfig personalizzati, ma non di crearne di nuovi. Per questo motivo è necessario creare il file di configurazione esternamente, testarlo

⁴gdb può essere utilizzato in modalità client/server per poter eseguire le fasi di debug su un normale computer host (gdb) per un binario in esecuzione su un dispositivo target (gdb-server).

⁵È necessario un collegamento ad internet.

ed in seguito aggiungerlo nella apposita sezione che verrà creata in un secondo momento.

La compilazione del kernel richiede la presenza del codice sorgente. Contrariamente a quanto si potrebbe pensare, non esiste un unico sorgente di riferimento. Come sappiamo il kernel Linux è rilasciato con licenza GNU GPLv2 dal sito ufficiale (kernel.org). Il kernel rilasciato da questo sito rappresenta la base di partenza da cui vengono elaborate tutte le altre versioni del kernel e, in ambito tecnico ci si riferisce ad esso con il termine “Vanilla”. Bisogna però considerare che esistono differenti versioni rilasciate contemporaneamente dal sito, molte delle quali sono definite “longterm”, ovvero che riceveranno aggiornamenti e correzioni per un tempo più lungo del classico ciclo di sviluppo.



Figura 5.2: Pagina del sito kernel.org

L'utilizzo di longterm è fondamentale in campo industriale, perché, dal punto di vista dello sviluppo di un prodotto, non sarebbe possibile seguire costantemente il continuo sviluppo del kernel e delle API. Per chiarire ulteriormente questo concetto, basti pensare che un sistema sviluppato su una longterm più datata potrebbe non funzionare perfettamente se eseguito con un kernel più recente e viceversa. In fase di sviluppo di un sistema ex-novo, come nel caso in questione, i kernel-headers (gli header file del sistema operativo) devono essere della medesima versione del kernel utilizzato.

Esistono quindi varie versioni marchiate come “longterm”, una versione definita come “last-stable” (l'ultima versione considerata stabile e testata, ma su cui non è garantito il mantenimento) e la versione “Mainline”, dove confluiscano tutti gli aggiornamenti e le novità.

Nel caso dei sistemi GNU/Linux Desktop, Workstation e Server, si parte da una versione Vanilla longterm, aggiungendo un numero arbitrario di patch. Ciò è in generale non valido nel mondo embedded, dove le aggiunte e gli adattamenti al kernel raramente confluiscano interamente nella Mainline. Ciò significa che il kernel vanilla, per quanto aggiornato, potrebbe non avere il necessario supporto al SoC utilizzato. Le motivazioni di tale mancato riversamento di codice sono per lo più da ricercarsi nell'uso di IP con forti limiti dal punto di vista contrattuale, dallo scarso interesse del produttore del SoC a mantenere aggiornati i driver o

per problemi di qualità dei driver considerata insufficiente dai manutentori del kernel mainline. Per sopperire a tale mancanza, i produttori dei SoC offrono una loro versione del codice sorgente del kernel Linux, spesso rilasciato con versioni considerate stabili al momento del rilascio del SoC stesso. C'è da considerare che tale codice sorgente non è sempre accessibile dagli utilizzatori finali, o se accessibile, non sempre è in forma completa (alcuni driver potrebbero essere stati rimossi per motivi di licenza). Infine, non è possibile trascurare il fatto che, tra il rilascio del SoC o della evalutaion board ed il suo utilizzo pratico nella realizzazione di un dispositivo finale, può passare un tempo non trascurabile (talvolta anche anni).

Nel progetto in cosiderazione, la versione stabile utilizzata da buildroot è una versione 3.12 modificata dal produttore, decisamente troppo datata per adeguarsi agli scopi del nostro progetto che richiede, tra le altre cose, il più ampio numero di driver per dispositivi wireless possibile, e con il miglior compromesso tra stabilità ed aggiornamento, oltre che la più recente versione del protocollo B.A.T.M.A.N. utilizzato per la realizzazione della rete mesh (2015.1). Un kernel troppo datato inoltre, pone delle forti preoccupazioni relative alla sicurezza, in particolar modo considerando l'uso del dispositivo all'interno di una rete condivisa con "peer" non necessariamente noti.

5.3.2 Download ed ambiente di compilazione

```
$ cd ~/mesh_system  
$ source toolchain.source  
$ mkdir kernel; cd kernel
```

In questi primi step viene creata all'interno della cartella del progetto una cartella di lavoro per lo sviluppo del kernel. La chiamata del comando "source" ci permette di richiamare le variabili di ambiente precedentemente salvate, senza le quali non sarebbe possibile cross-compilare il sorgente. Tale comando va eseguito solo una volta per ogni nuova sessione di terminale aperta. Alla chiusura di detta sessione, le variabili di ambiente non verranno salvate (comportamento desiderabile in un ambiente di sviluppo multiplo).

```
$ git clone https://github.com/beagleboard/linux.git  
$ cd linux
```

Con il comando git⁶ viene scaricato il codice sorgente del kernel (una versione community derivata da una release stable) con modifiche specifiche per questa evaluation board dal noto repository pubblico github. Questa versione non gode dello sviluppo e della fase di testing del kernel rilasciato ufficialmente con la board, ma è di gran lunga più recente ed adeguato alle esigenze progettuali.

L'utilizzo di una versione presa da un git pubblico, oltre ai naturali vantaggi derivati dall'uso di un SCM, faciliterà notevolmente l'opera di integrazione successiva con buildroot.

⁶Git è un software della categoria SCM sviluppato inizialmente da Linus Torvalds, l'autore del quasi omonimo kernel. Come altri software di questa fascia consente di controllare lo sviluppo del software, con una gestione avanzata su più rami di sviluppo. La sua natura volutamente distribuita, particolarmente adatta a codici sorgenti a sviluppo condiviso, lo ha reso la scelta predefinita nel mondo del software libero.

Una volta finita la fase di clone⁷ (assai più onerosa in termini di download rispetto al semplice pacchetto compresso), verrà creata una cartella nella quale è necessario entrare.

```
$ git checkout 94a60fd127ea564a483c577d277d68543b250c21 -b  
→ bbb_mesh
```

Con questo comando si imposta il sorgente in un punto particolare del suo sviluppo, e si crea un nuovo ramo (branch) locale per poter applicare tutte le eventuali modifiche e poter eseguire le eventuali operazioni di “versioning”. Tale modalità operativa risulta fondamentale durante una fase di sviluppo, perché consente di poter riesaminare in un secondo momento tutte le modifiche fatte, e la creazione di patch apposite per il progetto. Il “branch” originale è quello della versione longterm 4.1.

5.3.3 Configurazione

```
$ make bb.org_defconfig
```

Come già visto in precedenza, anche in questo caso, la base di partenza è un file defconfig. Questo specifico defconfig non si trova nella versione “mainline” del kernel, ma è una specifica aggiunta degli autori di questa versione relativa all’hardware utilizzato. Una volta eseguito il comando, il file di configurazione finale verrà salvato nella directory corrente nel file nascosto⁸ “.config”.

In questa versione, l’uso del defconfig così come impostato mal si presta all’uso pratico (l’immagine prodotta è di circa 7.88MB compressa, esageratamente sovradimensionata per le necessità progettuali). Inoltre, alcuni supporti necessari al corretto funzionamento di questo progetto non sono presenti. Per questi motivi è opportuno riconfigurare la configurazione del kernel.

```
$ make menuconfig
```

Anche in questo caso, come già visto in precedenza con buildroot, si presenta un menu sviluppato con ncurses per la selezione dei parametri di compilazione. In questo caso si tratta di un menu assai più complesso ed esaustivo, atto a configurare tutti gli aspetti del kernel.

Le possibilità nella scelta dei parametri di configurazione sono molteplici, risulta quindi difficile poter descrivere una politica di scelta omnicomprensiva. È possibile però definire dei comportamenti tipici sulla base delle scelte svolte durante questa fase.

Monolitico o Modulare⁹:

⁷Il comando git clone crea una copia locale del repository sul branch predefinito, solitamente etichettato “master”.

⁸La dicitura nascosta potrebbe essere fuorviante: nei sistemi Unix/Linux in genere, anteponendo il carattere “.” ad un nome file/cartella quest’ultimo non verrà visualizzato nelle viste relative a file (es. ls), a meno di opzioni. Ciò non ne impedisce l’accesso o la visualizzazione diretta.

⁹Tale terminologia può risultare ingannevole, in quanto è la medesima utilizzata per descrivere il modello architettonale del kernel. In questo contesto si riferisce semplicemente alle scelte relative alla compilazione dei moduli, ma Linux rimane un kernel basato su architettura monolitica (vedere sezione 4.1.1).

Monolitico: con questo termine si indica la tendenza a porre i driver e i supporti correlati all'interno dell'immagine del kernel, limitando o eliminando completamente l'uso dei moduli. Fino all'avvento del Linux Kernel 2.2, con l'introduzione dei moduli esterni, questa era l'unica scelta possibile. Nel settore embedded è generalmente preferita questa modalità, in quanto limita fortemente i problemi relativi al caricamento dei moduli. Come conseguenza ovvia di questa scelta, risultano le maggiori dimensioni dell'immagine del kernel.

Modulare: con "modulare" si intende invece la scelta di porre driver e supporti, per quanto possibile, all'esterno dell'immagine del kernel, sfruttando il meccanismo dei moduli caricabili dinamicamente. Tale metodologia, solitamente tipica per i sistemi desktop, offre una maggiore flessibilità nella fase di caricamento (un modulo viene caricato solo se necessario) oltre ad una minor dimensione dell'immagine del kernel, ma richiede alcune particolari attenzioni. In generale, non si può affermare che il kernel e il root-filesystem contenente i rispettivi moduli si trovino nella stessa area di storage; è importante evitare che il driver attivo all'uso da parte del kernel del root-filesystem non risieda nel filesystem stesso (una dipendenza circolare dalla quale ne risulterebbe un "kernel panic"). Alcuni supporti del kernel non sono comunque compilabili come moduli.

"Custom" o "General Purpose"

Custom: con questo termine si indica la scelta di inserire all'interno del kernel esclusivamente i supporti considerati essenziali all'hardware o ai protocolli previsti. Ciò produce un kernel più leggero, ma contemporaneamente poco flessibile all'aggiunta di nuovo hardware o alla richiesta di nuovi protocolli. Tale scelta è generalmente legata a contesti embedded, o comunque molto specializzati. Non è infrequente che questo modello vada di pari passo con una compilazione di tipo "monolitica".

General Purpose: modello antitetico al "custom", è il tipo di scelta che viene fatta per massimizzare la compatibilità del sistema con il maggior numero possibile di hardware e protocolli supportati. Tipico delle distribuzioni linux per sistemi desktop, questo tipo di impostazione viene spesso associata al modello di configurazione modulare.

In definitiva, analizzando i requisiti del progetto, è necessario utilizzare un approccio ibrido, in cui gli elementi fondamentali del sistema siano in un kernel monolitico, ma che disponga al contempo di un discreto numero di moduli per dei dispositivi esterni aggiunti all'hardware iniziale che potrebbero variare nel tempo. In fase di realizzazione di un prodotto finito, quindi con tutti i componenti stabilmente presenti all'interno del PCB (o quasi), si potrebbe passare ad una scelta definitivamente più monolitica, eccezion fatta per quei moduli che per motivi di design o di licenza (es. driver proprietari) è necessario lasciare esternamente (es. tipico approccio utilizzato negli smartphone Android).

Tra gli altri fattori da considerare, possono esserci alcuni limiti dovuti a cause esterne al kernel stesso: in questo progetto esiste un problema con la versione di bootloader utilizzata che impone una dimensione massima per l'immagine del kernel. Tale limite rende di fatto il defconfig predefinito inutilizzabile. Condizioni come queste costringono lo sviluppatore ad una maggiore attenzione alla dimensione dell'immagine finale, condizionando alcune scelte pratiche.

L'analisi dell'intero “.config” file del kernel richiederebbe una trattazione separata che va oltre quelli che sono gli interessi di questo progetto. In questa sede la discussione sarà limitata agli elementi di modifica sostanziale rispetto al “defconfig”, e sulle aggiunte relative alle necessità del progetto.

Le scelte fatte in questo specifico defconfig sono decisamente in controtendenza a quanto sottolineato sin ora: risultano aggiunti numerosi driver non necessari alla piattaforma base, sia come moduli esterni che compilati internamente.

In una prima fase, l'azione di configurazione è rivolta a rimuovere tutto il supporto per dispositivi non utili ai fini del progetto. Tale operazione richiede una notevole conoscenza del kernel, oltre che della dotazione hardware della board utilizzata (visibile nella sezione 3.5.4). In una seconda fase di ottimizzazione è inoltre possibile rimuovere anche il supporto per hardware presente sulla board, ma non realmente utilizzato nel progetto.

Come approccio intermedio, per quei driver “transitori” (per hardware che potrebbe o potrebbe non essere incluso nel progetto finale) ma utili in fase di sviluppo, è preferibile, laddove possibile, la compilazione come modulo.

Il defconfig originale prevede il caricamento di alcuni firmware, sotto forma di blob¹⁰ binari all'interno dell'immagine principale. Tale scelta, pur semplificando in parte il lavoro dello sviluppatore, risulta particolarmente pesante dal punto di vista delle dimensioni. In particolare essendo un defconfig generico per più board differenti non tutti sono da considerarsi necessari.

Nel menuconfig questa voce si trova sotto i menu “Device Drivers → Generic Driver Options → Include in-kernel firmware blobs in kernel binary”, o all'interno del file di configurazione con le seguenti voci

```
CONFIG_FIRMWARE_IN_KERNEL=y
CONFIG_EXTRA_FIRMWARE="am335x-pm-firmware.elf am335x-bone-scale-
    ↲ data.bin am335x-evm-scale-data.bin am43x-evm-scale-data.bin"
CONFIG_EXTRA_FIRMWARE_DIR="firmware"
```

Nello specifico, il firmware am43x-evm-scale-data.bin risulta inutile per questa piattaforma. Non è però possibile eliminare il più grande di questi firmware, il file “am335x-pm-firmware.elf”, in quanto fondamentale per la gestione energetica del dispositivo.

Altro passaggio chiave della configurazione del kernel per questo progetto è la abilitazione del modulo BATMAN nel menu “Networking Support → Networking Options → B.A.T.M.A.N. Advanced Meshing Protocol”.

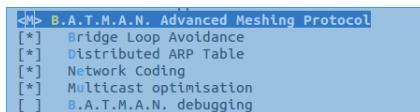


Figura 5.3: Moduli kernel per B.A.T.M.A.N.

All'interno del file di configurazione sono visibili queste voci.

```
CONFIG_BATMAN_ADV=m
CONFIG_BATMAN_ADV_BLA=y
```

¹⁰Tale termine è in genere riferito a tutti quei firmware integrati all'interno di un modulo kernel o caricati successivamente che, a discapito della natura opensource del driver, non sempre sono rilasciati sotto forma di sorgente. Tale approccio è alquanto comune nel caso di schede grafiche, wireless, bluetooth ed altri driver proprietari.

```
CONFIG_BATMAN_ADV_DAT=y  
CONFIG_BATMAN_ADV_NC=y  
CONFIG_BATMAN_ADV_MCAST=y
```

In particolare, oltre al modulo di BATMAN, è possibile impostare il supporto per alcune feature di BATMAN viste nella sezione 2.2.1.

Infine è necessario abilitare la compilazione dei driver per l'hardware selezionato. Il modulo per il dispositivo wifi nel menu “Device Drivers → Network Device Support → Wireless LAN → Realtek 8187 and 8187B USB support” o all'interno del file di configurazione.

```
CONFIG RTL8187=m  
CONFIG RTL8187_LEDS=y
```

Il modulo bluetooth si trova invece nel menu “Networking support → Bluetooth Subsystem Support → Bluetooth Device Drivers → HCI USB Driver” o all'interno del file di configurazione alla voce.

```
CONFIG_BT_HCIBTUSB=m  
CONFIG_BT_HCIBTUSB_BCM=y
```

Entrambi i driver necessitano che i rispettivi sottosistemi siano abilitati, compilati e avviati al momento del caricamento del modulo. Per poter usufruire di alcuni tool a riga di comando per la gestione delle schede wifi è inoltre necessario abilitare la “wireless extensions compatibility”.

```
CONFIG CFG80211_WEXT=y
```

Tutte le altre numerose modifiche al defconfig originale, per quanto abbiano una fattiva influenza sull'immagine risultante, non sono di stretto interesse per questo progetto, quindi di scarso interesse in questo contesto.

5.3.4 Compilazione

```
$ make uImage LOADADDR=80008000
```

Con questo comando si avvierà la compilazione del kernel (ma non dei moduli) specificando il tipo d'immagine da utilizzare ed eventuali parametri. L'immagine prodotta non sarà quella utilizzata in fase di rilascio, ma sarà utile per la fase di testing iniziale e di creazione del “defconfig” definitivo.

Il LOADADDR è un valore assai importante, che indica il punto in memoria da cui il kernel inizierà la sua esecuzione. Tradizionalemente è posto 32Kb (0x8000) al di sopra della prima area di memoria disponibile, per lasciare spazio per l'inserimento dei parametri ATAGs¹¹. In generale è raro trovarsi a calcolare il LOADADDR, ed è di solito fornito dal produttore del SoC o della Board nelle guide di riferimento o negli esempi di configurazione di U-boot.

```
$ make modules  
$ make modules_install INSTALL_MOD_PATH=$TEMP_ROOTFS
```

Con questa fase si lancia la compilazione dei moduli, e successivamente l'installazione dei moduli compilati in un percorso specificato dalla variabile

¹¹Simili a variabili di ambiente, passate dal bootloader al kernel. Tale sistema non è universalmente supportato.

“TEMP_ROOTFS”¹². TEMP_ROOTFS è in realtà una variabile d’ambiente preimposta contenente il percorso dell’ambiente di test (vedere sezione A.1).

Le immagini del kernel

Si possono suddividere le tipologie d’immagine del kernel in base alle informazioni per specifici bootloader o al tipo di compressione utilizzata.

Image: immagine binaria non compressa del kernel. L’uso di questo tipo di soluzione è scarsamente raccomandabile se non per motivi di debug. Il bootloader non è in grado di sfruttare le massime potenzialità del sistema, ciò comporta che la maggiore dimensione dovuta alla mancata compressione impatterebbe sensibilmente sui tempi di avvio. Nel caso di dispositivi embedded di piccole dimensioni ciò è da escludere completamente.

zImage: immagine binaria compressa autoestrante¹³ con algoritmo gzip. zImage è utilizzato come scelta di default per molti sistemi embedded: sebbene gzip non garantisca i medesimi risultati di altri software di compressione, risulta computazionalmente semplice, e quindi non pesante dal punto di vista dei tempi di esecuzione.

bzImage: medesimo meccanismo di zImage, ma con algoritmo bzip2: più efficace dal punto di vista della compressione, risulta più oneroso computazionalmente. È stata la scelta di default per molti sistemi desktop/server in molte distribuzioni.

uImage: uImage è il formato predefinito per le immagini kernel da utilizzare con il bootloader u-boot (vedere la sezione U-boot image formats 4.4.1). Di per sé non specifica il tipo di compressione, ma aggiunge un header di informazioni all’immagine utili allo stesso u-boot per il corrente caricamento ed esecuzione dell’immagine. In base al makefile, questo tipo di immagine è solitamente compresso in gzip.

Nell’ottica di avere un’immagine kernel funzionante, con i relativi moduli, possono essere necessarie numerose ripetizioni della procedura, con un numero cosiderevole di prove nell’ambiente di test. Ottenuta un’immagine funzionante si può passare alla costruzione del rootfsystem.

Nella fase iniziale di configurazione dell’ambiente sono già stati impostati i parametri fondamentali del sistema (compilatore, libreria C, impostazioni di architettura). In seguito è necessario specificare tutti gli altri parametri integrando la versione del kernel appena testata.

Anche in questa fase è necessario operare un’accurata selezione sui pacchetti a disposizione per il sistema: nella maggior parte dei casi, lo spazio a disposizione per lo storage risulta essere un parametro fondamentale quanto i limiti computazionali del dispositivo.

¹²Senza il parametro INSTALL_MOD_PATH i moduli non verrebbero installati nel sistema target ma in quello host, che essendo un’architettura differente potrebbe subirne un danno dal punto di vista OS.

¹³I file compressi autoestratti sono da considerarsi dei file che al momento dell’esecuzione decomprimono il loro contenuto in un’area di memoria.

5.4 Creazione del Rootfs

In questa sezione verranno descritti i passaggi necessari per la realizzazione delle immagini finali del rootfilesystem.

5.4.1 Buildroot Board Directory

All'interno di buildroot, oltre ai già citati “defconfig”, sono presenti alcune cartelle specifiche per ogni differente piattaforma supportata. Si trovano tutte all'interno della cartella “board”, e contengono alcuni elementi aggiuntivi e personalizzabili, non presenti nella normale configurazione.

```
cd buildroot-2015.08.1/board/
mkdir meshsystem
cp ../../kernel/linux/.config meshsystem/linux-4.1.2.config
cp ../../uEnv.txt meshsystem/uEnv.txt
cp ../../post-image.sh meshsystem/post-image.sh
```

Nei primi due passaggi ci si sposta all'interno della cartella “board” di buildroot e si crea una cartella appositamente per il sistema in sviluppo (nominata “meshsystem”). A seguire vengono copiati tre file al proprio interno:

linux-4.1.2.config: il “.config” del file preparato nella sezione precedente e pronto per essere utilizzato.

uEnv.txt: in questo file è possibile specificare alcune opzioni e variabili di ambiente di u-boot (vedere sezione 5.6)

post-image.sh: questo script fa eseguire alcune operazioni a buildroot, dopo la creazione delle immagini di sistema (vedere sezione 5.4.4)

5.4.2 Aggiunta di un pacchetto esterno

Buildroot consente l'installazione di pacchetti aggiuntivi da essere compilati ed installati all'interno del rootfilesystem finale. Per fare ciò è necessario che il software in questione sia rilasciato sotto forma di sorgente (in casi eccezionali come binario) e che sia disponibile in forma di pacchetto compresso o di repository git/svn/mercurial/cvs.

Nel caso specifico per semplificare, è possibile utilizzare un pacchetto compresso in formato tar.gz.

In questa fase, è necessario aggiungere il software per il protocollo MoM (illustrato nella sezione 6.3) creato appositamente per questo progetto, e che verrà qui utilizzato come esempio di personalizzazione di buildroot.

Per aggiungere il pacchetto è necessario seguire i seguenti passi:

```
$ cd buildroot-2015.08.1/package
$ mkdir mom
$ cd mom
$ touch Config.in mom.hash mom.mk
```

Il contenuto del primo file, ”Config.in“, può essere all'incirca questo:

```
config BR2_PACKAGE_MOM
    bool "mom"
    help
```

```
This is a Client/Server utility designed to work over
→ B.A.T.M.A.N. mesh network
```

Questo file serve per l'integrazione del nostro software all'interno del menu di configurazione di buildroot. Questo passo non è di per sé sufficiente, in quanto, per essere visibile dovrà essere aggiunto anche nel file di configurazione (al solito Config.in) posizionato nella cartella "package".

```
menu "Networking applications"
    package/mom/Config.in
```

Così come presentato è da considerarsi solo un esempio. In genere i pacchetti vanno aggiunti in ordine alfabetico, ed in questa sezione sono molti i pacchetti che lo precedono.

Il secondo file è un file di hash¹⁴ utilizzato per verificare la correttezza del pacchetto scaricato dalla rete, verificandone quindi sia l'integrità sia eventuali manomissioni.

Per la creazione di un file di hash è sufficiente utilizzare il seguente comando nel percorso in cui è presente il pacchetto compresso.

```
$ sha256sum mom-0.1.tar.gz > mom.hash
$ cat mom.hash
sha256      502
→ fb74053b02548c175d5d5b04dcdaaa33422571f92ecb95564b7ad144c4c8d
→     mom-0.1.tar.gz
```

Il secondo comando è necessario solo per la verifica della corretta creazione dell'hash.

Il terzo file è quello che descrive la modalità di compilazione e installazione effettiva del software all'interno del futuro filesystem. Un'analisi accurata consente di comprenderne al meglio i valori.

```
MOM_VERSION=0.1
MOM_SOURCE=resist-$(RESIST_VERSION).tar.gz
MOM_DEPENDENCIES=
MOM_SITE = http://los-it.net/releases/
MOM_LICENSE= GPLv2

define MOM_BUILD_CMDS
    $(TARGET_CONFIGURE_OPTS) $(MAKE) ARCH=$(KERNEL_ARCH)
        → CROSS_COMPILE=$(TARGET_CROSS) -C $(@D)
endef

define MOM_INSTALL_TARGET_CMDS
    $(INSTALL) -D -m 0755 $(@D)/mom_client $(TARGET_DIR)/usr/
        → bin
    $(INSTALL) -D -m 0755 $(@D)/mom_server $(TARGET_DIR)/usr/
        → bin
    $(INSTALL) -D -m 0755 $(@D)/libmom.so $(TARGET_DIR)/usr/lib
endef
$(eval $(generic-package))
```

Tutte le variabili che iniziano con il prefisso "MOM_“, ovvero con il nome del pacchetto, sono state definite appositamente per la compilazione di questo

¹⁴I file di "hash" prendono il nome dalle funzioni di hash utilizzate per la loro creazione. Per semplificare, un hash è un identificativo alfanumerico a lunghezza prefissata, ottenuto come funzione di un algoritmo e di un file di lunghezza arbitraria.

software. Sul piano pratico, le altre variabili di ambiente permettono di poter trascurare, almeno in parte, i problemi dovuti alla cross-compilazione e all'installazione all'interno dell'immagine finale.

In questo caso, trattandosi di un Makefile relativamente semplificato (ma che deve assolutamente prevedere l'uso di un cross-compiler, sfruttando la variabile di ambiente CROSS_COMPILE) è stato necessario specificare tutti i singoli passaggi. In presenza di pacchetti che sfruttano sistemi di autoconfigurazione del codice (es. autotools), la situazione è assai semplificata.

Un analogo lavoro dovrà essere svolto per l'altro software creato per questo sistema (mesh-init).

5.4.3 Configurazione Buildroot

Una volta creata la cartella board, e quelle relative ai pacchetti aggiuntivi è possibile passare alla fase di configurazione.

Verranno evidenziati alcuni passi significativi relativi alla configurazione.

Cambiamento porta seriale

Nella sezione "System Configuration → getty options → TTY port" viene definita quale sia la porta (virtuale o fisica) sulla quale mettersi in ascolto per il terminale principale (il primo che parte all'avvio). Tale scelta in realtà ha un valore prevalentemente nella fase di debug del sistema, in quanto, in molti prodotti finiti tale terminale viene disabilitato. Nel caso della board, il defconfig fa riferimento alla prima porta seriale, con il nome "ttyO0". Tale nome, era valido nella versione di kernel 3.12, quella utilizzata dal defconfig, ma non più valido con la versione scelta, dove la stessa porta prende il nome di ttyS0. Il kernel provvede in fase di caricamento, tramite un apposito modulo, a convertire tutte le chiamate a "ttyO0" in "ttyS0"; tale conversione però non avviene più dopo l'avvio di init, rendendo necessaria tale correzione.

Kernel Linux

In questa parte del menu di configurazione è possibile utilizzare quanto sviluppato in precedenza, utilizzando gli stessi elementi testati separatamente all'interno di buildroot.

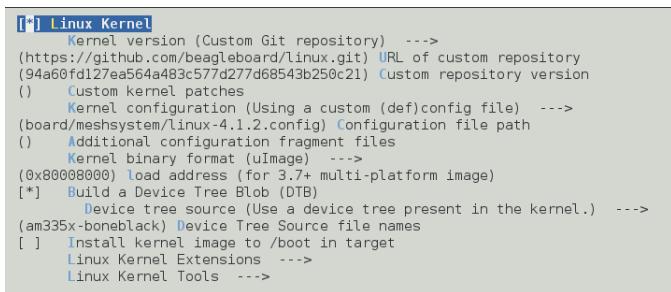


Figura 5.4: Menu di configurazione del kernel in buildroot

Come evidenziato dalla figura 5.4, si utilizza il git visto in precedenza, facendo puntare il sorgente al medesimo ID di commit¹⁵ con il file di configurazione precedentemente sviluppato e testato, aggiunto alla cartella board, e con il medesimo LOADADDR. Tra le opzioni è possibile anche specificare il "device-tree" (vedere sezione 5.1.2) da utilizzare. Nel defconfig originale utilizza il tipo di immagine "zImage" che, pur essendo supportato dalle odierne versioni di u-boot, può risultare una scelta meno saggia rispetto al formato nativo di u-boot (uImage), scelto in questa configurazione.

Filesystem Images

In questa sezione è possibile specificare il tipo d'immagine prodotta da buildroot come risultato della compilazione. Oltre al file compresso utilizzato nell'ambiente di test, è necessario aggiungere un'immagine del rootfs sotto forma di filesystem specificando i seguenti parametri

ext2/3/4 variant (ext4): specifica il tipo di filesystem utilizzato, nello specifico ext4¹⁶

filesystem label (meshsystem): etichetta del filesystem.

Size in block (65536): la dimensione in blocchi definisce quella che sarà la dimensione finale del filesystem. Il filesystem di per sé richiederebbe meno spazio, ma può risultare saggio lasciare un po' di spazio aggiuntivo per altri scopi.

Compression method (gzip): anche se il filesystem verrà comunque utilizzato in forma decompressa, per motivi di caricamento, avere tale file compresso può tornare vantaggioso sia in tempi di caricamento, sia in spazio occupato sul dispositivo di storage.

Target Packages

La scelta dei pacchetti da installare è di fatto una delle prime discriminanti nel determinare la dimensione finale del rootfs finale. In questo caso è importante limitare il più possibile il numero di pacchetti installati, senza però sacrificare la funzionalità prevista.

gpsd (/dev/ttyRFCOMM0): il pacchetto gpsd consente un facile interfacciamento con i dispositivi GPS. Risulta necessaria specificare la porta di comunicazione utilizzata, in questo caso si tratta dell'astrazione del dispositivo RFCOMM, utilizzata genericamente per i dispositivi bluetooth con un comportamento assimilabile ad una porta seriale.

Bluez: implementazione dello stack bluetooth per Linux.

batctl: tool per la configurazione delle reti B.A.T.M.A.N. (vedere sezione B.3)

¹⁵Ogni commit all'interno del medesimo git ha un identificativo unico (commit id) composto da cifre e lettere.

¹⁶ext4 è il filesystem utilizzato come scelta predefinita dalle maggiori distribuzioni GNU/Linux. In campo embedded il suo utilizzo è ancora molto discusso, in quanto, a fronte di una grande potenza e affidabilità, richiede un notevole costo operazionale poco compatibile con dispositivi di memoria quali NAND/FLASH/MMC.

iproute2: tool per la configurazione del routing.

iptables: tool di configurazione del firewall.

iputils: un insieme di tool per la configurazione dei dispositivi di rete tramite linea di comando.

iw: tool per la gestione di schede wireless (mac80211).

Wireless Tools: una serie di tool per la configurazione delle schede wireless (cfg80211)

wpa_supplicant: tool per l'accesso a reti WiFi protette da sistema WPA¹⁷

MoM: il tool per la comunicazione attraverso rete mesh ideato per questo progetto, aggiunto a buildroot

Mesh-Init: il tool per la configurazione semplificata del sistema all'avvio.

Ultimate le modifiche è necessario uscire dal "menuconfig" salvando il file di configurazione. Il file di configurazione finale (.config), verrà copiato all'interno della cartella "config" in modo tale da diventare a sua volta un defconfig riutilizzabile automaticamente per ogni eventuale modifica e personalizzazione.

```
$ cp .config configs/meshsystem_defconfig
```

5.4.4 Compilazione del rootfs

Arrivati ad una configurazione definitiva per il rootfs, è necessario ripetere i passi compiuti in precedenza.

```
$ make clean  
$ make source  
$ make
```

Una volta ultimata questa fase, all'interno della cartella "out/images/" saranno presenti i seguenti file

- uboot.img
- uImage
- rootfs.tar.gz
- rootfs.ext2.gz
- MLO
- uEnv.txt

Tutti i file presenti, eccezione fatta per il file rootfs.tar.gz (creato solo per l'ambiente di test), andranno a comporre il dispositivo di storage di boot del sistema.

¹⁷Wi-Fi Protected Access (WPA e WPA2): un insieme di protocolli atti alla sicurezza e alla autenticazione nelle reti wireless.

5.5 Dispositivo di memoria

Le scelte per il dispositivo di memoria possono essere molteplici, e possono variare molto sulla base del tipo e delle capacità effettive di memoria a disposizione, dal tipo di uso che se ne vuole fare e dal tipo di kernel e bootloader in utilizzo.

Dovendo essere un dispositivo in grado di avviarsi in modo indipendente, risultano possibili ma assolutamente poco pratici i sistemi di avvio attraverso la rete (TFTP¹⁸/NFS¹⁹).

L'utilizzo di una EEPROM, risulta essere limitante sia dal punto di vista dello spazio che del costo complessivo. Inoltre, non è sempre possibile installarle.

Nella passata generazione di dispositivi embedded, quelli dotati di memorie NOR/NAND la scelta era assai limitata, sia per le dimensioni dello spazio a disposizione, sia per l'impossibilità di utilizzare alcuni tipi di filesystem. I filesystem adeguati per l'uso con questo tipo di memorie sono:

jffs1/2: un "Log Structured Filesystem" disegnato per lavorare su NAND, ha alcuni limiti strutturali legati alla necessità di eseguire le fasi di clean ogni qual volta montato.

yaffs1/2: concettualmente simile ai filesystem JFFS, con il quale condivide alcuni limiti. Molto utilizzato dai produttori dei primi smartphone Android.

ubifs: di recente adozione, Ubifs introduce molte ottimizzazioni legate al "wear-leveling" e con minori tempi di accesso in fase di mount.

La scelta di una memoria NAND classica per dispositivi embedded di fascia alta può risultare ad oggi anacronistica. Con l'avvento delle ultime generazioni di dispositivi embedded, si è passati a memorie più efficaci, più capienti e dotate di un maggiore livello di astrazione (ad esempio le attuali eMMC), tali da consentire una maggiore libertà sulla scelta del filesystem da utilizzare.

In alcuni dispositivi, per motivi di prezzo e semplicità implementativa, si è passati direttamente all'uso di memorie flash esterne ad alta capienza (come SD card e USB key).

La board ha a disposizione sia una eMMC interna (astratta come una SD card) sia uno slot per schede MicroSD.

Nell'ottica di realizzare un dispositivo economico, flessibile ed affidabile, facilmente configurabile ed aggiornabile, è stata scelta la possibilità di utilizzare la MicroSD esterna.

Posto di utilizzare la scheda esterna, esistono svariati scenari di configurazione, ma per semplicità espositiva verranno presentati solo i due più significativi: quello definito come "rootfs su partizione" e l'utilizzo di immagini precaricate, entrambi basati su un sistema di partizionamento classico.

5.5.1 Il partizionamento classico

Con questo termine si indica l'utilizzo di una schema di partizionamento tradizionale (MSDOS Partition Table²⁰) in cui gli elementi del rootfilesystem si trovano all'interno di un classico filesystem collocato in una delle partizioni. Lo schema di

¹⁸Protocollo semplificato di File Transfer Protocol, ideato per il boot[10].

¹⁹Network filesystem: filesystem utilizzabile attraverso connessione di rete[9].

²⁰Questo schema si basa sulla presenza nel dispositivo di storage di una struttura dati posta all'inizio dello spazio disponibile che prende il nome di Master Boot Record (MBR).

partizionamento MSDOS soffre di numerosi limiti, sia per numero di partizioni che per spazio allocabile, ma questi non sono particolarmente significativi in ambito embedded, in quanto si tratta generalmente di dispositivi di memoria dalla ridotta capacità e che non necessitano un grande numero di partizioni.

Pur esistono da tempo uno nuovo schema di partizionamento, il GPT, (Guid Partition Table) quest'ultimo non è ancora universalmente supportato all'interno dei bootloader, rendendone di fatto impraticabile o poco probabile l'utilizzo in ambito embedded.

Rootfs su partizione

In questa modalità il filesystem contenente il rootfs viene disposto direttamente su una partizione classica. Trattandosi di un sistema GNU/Linux, il tipo di partizione prende il nome "Linux" (codice 0x83) ed il tipo di filesystem è di tipo ext2/ext4. Per motivi di compatibilità con i filesystem della famiglia ext con il bootROM o in generale con i vari bootloader, è assai probabile che non sia possibile fare la medesima scelta per la partizione che contiene i file del pre-bootloader, del bootloader e del kernel Linux, dovendo optare per un filesystem di tipo FAT (codice 0x0C) formattato in FAT32. Prendendo ad esempio il dispositivo utilizzato nel progetto (per comodità chiamato con il medesimo nome utilizzato dal kernel per indicare il dispositivo, ovvero "mmcblk0") un possibile schema di partizionamento potrebbe essere il seguente:

mmcblk0: dispositivo

mmcblk0p1 (64MB): tipo 0x0C, FAT32, etichetta "BOOT". Contiene i file immagine del pre-bootloader, del bootlooter e del kernel, oltre al file con le variabili d'ambiente per il bootloader.

mmcblk0p2 (512MB): tipo 0x83, ext2/4, etichetta rootfs. Contiene i file di tutto il sistema userspace oltre che i moduli del kernel

mmcblk0p3 (1400MB): partizione opzionale per i dati da salvare o per altre informazioni. Per esigenze particolari tale spazio può essere ridotto in favore del rootfs. Può essere formattato sia in FAT che in ext.

I pro di tale scelta sono:

- Semplicità d'implementazione: si tratta del modo più intuitivo di disporre il rootfs.
- Semplicità di configurazione: il bootloader è già configurato per sfruttare questo schema
- Possibilità di sfruttare tutto lo spazio all'interno della scheda per il rootfs; di conseguenza aumenta il numero di software nel sistema.
- Possibilità di aggiornare in modo permanente il contenuto del rootfs durante l'esecuzione del sistema

Tale struttura è della dimensione di 512 byte, di cui i primi 446 riservati alla fase 1 del bootloader, 64 byte riservati alle informazioni sulle partizioni e 2 byte finali chiamati MAGIC NUMBER come campo identificativo. Questo schema consente l'uso di 4 partizioni primarie o di 3 partizioni primarie e 11 partizioni logiche.

I contro di tale scelta sono:

- La copia o l'aggiornamento dei file del rootfs all'interno del dispositivo possono risultare complessi se non utilizzati con un sistema Linux (Windows e MacOS hanno problemi ad interagire con i filesystem della famiglia ext).
- I filesystem della famiglia ext sono pensati e ottimizzati per i dischi rotazionali; alcune di queste ottimizzazioni possono portare ad un maggior deterioramento del dispositivo.
- In caso di spegnimento improvviso del sistema o di interruzione di corrente, si possono verificare perdite di dati, o problemi in fase di riavvio del dispositivo.
- Possibilità di aggiornare in modo permanente il contenuto del rootfs durante l'esecuzione del sistema.

Immagini precaricate

Questo schema, pur utilizzando il partizionamento classico come base, anche per le sopracitate esigenze del bootROM e del bootloader, non lo adotta per il rootfs. In pratica tutti gli elementi del sistema sono salvati come singoli file immagine compressi, compreso il rootfs. Utilizzando la medesima convenzione utilizzata nella precedente sezione è possibile rappresentare un potenziale schema di partizionamento.

mmcblk0: Dispositivo

mmcblk0p1 (512MB): tipo 0x0C, FAT32, etichetta BOOT. Contiene tutte le immagini di sistema in formato compresso oltre al file con le variabili di ambiente per il bootloader.

mmcblk0p2 (1400MB): partizione opzionale per i dati da salvare o per altre informazioni. Per esigenze tale spazio può essere ridotto in favore del rootfs. Può essere formattato sia in FAT che in ext.

I pro di tale scelta sono:

- Ridotto tempo di accesso I/O in fase di esecuzione
- Estrema facilità di copia ed aggiornamento dell'intero sistema partendo da immagini pronte su qualsiasi sistema operativo (basandosi sull'elevata compatibilità del filesystem FAT)
- Ridotto numero di operazioni sul filesystem che contiene le immagini (opzionalmente utilizzabile anche in sola lettura), limitando le possibilità di guasto e di perdita di dati in caso di problemi.
- Impossibilità di aggiornare in modo permanente il contenuto del rootfs durante l'esecuzione del sistema.

I contro di tale scelta sono:

- Le dimensioni complessive del filesystem sono fortemente condizionate dalle dimensioni della RAM
- Una maggiore complessità di configurazione del bootloader
- Un tempo di avvio aumentato dovuto al caricamento dell'immagine del rootfs nella RAM, ed un conseguente consumo statico della RAM dovuto alle dimensioni del rootfs.
- Impossibilità di aggiornare in modo permanente il contenuto del rootfs durante l'esecuzione del sistema.

Si noterà come la possibilità o meno di poter modificare il contenuto del rootfs sia inteso sia come fattore "pro" che "contro": tale possibilità infatti permette di aggiornare o aggiungere software e configurazioni utili al sistema, rendendolo di fatto più flessibile, ma al contempo rende possibile anche modifiche permanenti non desiderabili all'interno del sistema, come ad esempio codice malevolo o errori all'interno dei file di configurazione. Nel caso del rootfs su partizione, una qual volta compromesso il filesystem rimane come unica soluzione il ripristino di rootfs stesso. Nel caso delle immagini precaricate potrebbe essere bastevole un riavvio di sistema, con conseguente ricaricamento dell'immagine originale.

Nel caso d'immagini precaricate c'è inoltre da considerare la possibilità di ridurre sensibilmente l'utilizzo di I/O sulla memoria, con effetti positivi sia sulla longevità del dispositivo che nei consumi energetici.

Minori (o quasi nulli) accessi I/O si traducono anche in una maggiore reattività del sistema. Per come è impostato al momento il sistema di init, sarebbe possibile rimuovere la sdcard una volta terminata la fase di boot e configurazione.

In questo progetto è stata scelta la modalità con immagine precaricate: non avendo bisogno di particolare spazio per il rootfs, disponendo di un discreto quantitativo di RAM e dovendo ottimizzare i consumi preservando la sicurezza e l'affidabilità che deve essere tipica di un dispositivo mobile progettato per essere connesso costantemente a reti differenti l'uso di immagine precaricate è risultata essere la scelta più congrua.

5.5.2 La configurazione della scheda SD, e la copia dei file

Le operazioni di creazione della scheda iniziale deve essere eseguita dal computer "host" attraverso una shell con permessi di root. In questa serie di comandi si assume che il device di riferimento su sistema host sia "mmcblk0" e che il dispositivo sia vuoto.

```
# fdisk /dev/mmcblk0
(n p 1 [Enter] +512MB)
(n p 2 [Enter] +1400MB)
(t 1 0C)
(t 2 0C)
(w q)
```

La sequenza di comandi è da intendersi intervallata dalla pressione del tasto invio tra ogni singola istruzione (ogni singola lettera è un comando). Laddove presente [Enter], indica di lasciare il parametrazione preimpostato.

La prima sequenza richiede al software fdisk di creare una nuova partizione (n), primaria (p), collocata al primo slot disponibile (1), partendo dal primo settore disponibile ([Enter]) e con una dimensione di 512MB (+512MB). La seconda sequenza esegue le medesime operazioni per la seconda partizione (di dimensioni maggiori).

La terza sequenza imposta il tipo di partizione (t) per la partizione numero 1 (1) e la imposta a 0C (FAT). Analoga situazione per la sequenza successiva riferita alla seconda partizione. Infine, l'ultima sequenza scrive la tabella delle partizioni (w) ed esce dal programma (q).

Una volta creata la tabella delle partizioni è necessario formattarle:

```
# mkdosfs -F 32 /dev/mmcblk0p1
# mkdosfs -F 32 /dev/mmcblk0p2
```

Ultimata la formattazione è ora possibile copiare le immagini all'interno del dispositivo di memoria.

```
# mkdir /mnt/meshdisk
# mount /dev/mmcblk0p1 /mnt/meshdisk
# cd $BUILDROOT/output/images/
# cp MLO uboot.img uEnv.txt uImage rootfs.ext2.gz /mnt/meshdisk
# sync
# umount /mnt/meshdisk
```

Queste ultime istruzioni, montano la scheda SD all'interno del sistema "host" e copiano le immagini come se fossero semplici file. In seguito, dopo un'operazione di "sync" (puramente precauzionale) verrà smontato il dispositivo di memoria al fine di essere reinserito all'interno della board per l'avvio di sistema.

5.6 uEnv.txt

Durante questa trattazione si è fatto più volte riferimento alle variabili di ambiente u-Boot ed al file uEnv.txt, ovvero il file di testo dove è possibile impostare le variabili di ambiente necessarie.

Queste variabili, e di conseguenza il file uEnv.txt che le contiene, possono essere soggette ad una grande varietà di possibili configurazioni, in base al tipo di avvio desiderato. Per semplicità si mostrerà solamente quella specifica di questo progetto, relativa al boot delle immagini finali del sistema.

```
meshargs=setenv bootargs console=tty00,115200n8 root=/dev/ram0 rw
    ↳ ramdisk_size=196608 initrd=0x81000000,64M
meshcmd=run meshargs; fatload mmc 0:1 0x81000000 rootfs.ext2.gz;
    ↳ fatload mmc 0:1 0x80200000 uImage; fatload mmc 0:1 0
    ↳ x80F80000 am335x-boneblack.dtb; bootm 0x80200000 - 0
    ↳ x80F80000
```

Analizzandone il contenuto, si può evincere che si tratti della definizione di due variabili d'ambiente, che a loro volta definiscono due distinte azioni da eseguire.

La prima, "meshargs" imposta a sua volta un'altra variabile, nota come bootargs; tale variabile è vitale per l'avvio del sistema, in quanto definisce i parametri che verranno passati al kernel.

console=tty00,115200n8: definisce quale sia la porta seriale e i suoi parametri di velocità e parità, sul quale verrà mostrato il caricamento del kernel.

Fondamentale durante la fase di test e sviluppo, ininfluente durante l'esecuzione sul prodotto finale

root=/dev/ram0: definisce quale sarà il dispositivo dal quale dovrà essere montato il rootfs; in questo caso si tratta di un dispositivo virtuale definito ramdisk sul quale verrà eseguita una mappatura dell'immagine del filesystem.

rw: indica che il dispositivo del rootfs verrà montato in modalità lettura/scrittura (read/write). Un parametro piuttosto insolito in sistemi normali, ma necessario per questa specifica configurazione.

initrd=0x81000000,64M: questo parametro indica l'indirizzo fisico dove verrà caricata l'immagine del rootfs.

La seconda variabile descrive il comportamento di uno script, ovvero di una serie di comandi da eseguire, fino al boot.

run meshargs: esegue la variabile definita in precedenza, impostando fattivamente i valori di bootargs.

fatload mmc 0:1 ...: queste tre chiamate carichano nell'ordine il rootfs, la uImage (kernel) e il device tree a differenti indirizzi fisici di memoria. Tali indirizzi non sono casuali, e devono essere compatibili con l'hardware in uso²¹. Il nome fatload indica che il caricamento avviene da una partizione fat, e l'opzione "mmc 0:1" indica che si sta utilizzando il primo dispositivo mmc (nel nostro caso la microSD) alla prima partizione.

5.7 mesh-init

Come già notato in precedenza, il dispositivo finale, almeno nella sua implementazione base, non dispone di dispositivi di I/O all'infuori dei tasti e dei led dedicati alla segnalazione (come ad esempio nello scenario di utilizzo "Sicurezza Personale" nella sezione 7.3). Per configurare il dispositivo con i giusti parametri è quindi necessario utilizzare un metodo di configurazione che sia contestualmente semplice dal punto di vista dell'utente comune e contemporaneamente non interattivo.

Per questo motivo è stato creato un particolare script, collegato ad init (vedere sezione 4.3.2) che, tramite la lettura di uno specifico file di configurazione (meshconf.txt), imposta i vari parametri di sistema e di rete.

Questa sistema semplifica la configurazione da parte dell'utente, che deve solamente creare o modificare il file di configurazione memorizzato all'interno della scheda di memoria. Considerando i molteplici campi di utilizzo, si potrebbe semplificare maggiormente la configurazione da parte del semplice utente fornendo a mezzo internet, direttamente un set di file di configurazione per contesti differenziati.

Il file di configurazione viene letto solamente durante la fase di inizializzazione del sistema, e rimane copiato in memoria sino al successivo riavvio.

²¹Diversamente da quanto si potrebbe credere, la mappatura della memoria non parte sempre dall'indirizzo 0x00000000.

Tale scelta è stata fatta per rendere possibile la rimozione della scheda SD una qual volta avviato il sistema. Un esempio di file di configurazione con la descrizione dei parametri è disponibile alla sezione B.2

Alexjan Carraturo

Capitolo 6

MoM

Al fine di semplificare il lavoro di sviluppo relativo ai possibili scenari di utilizzo del dispositivo dotato del sistema realizzato, è stato creato un protocollo di rete: MoM (Message over Mesh).

Con il termine protocollo di rete si intende un protocollo particolare di comunicazione ideato per il funzionamento in una rete informatica, ovvero come definizione formale delle modalità di interazione che due o più architetture elettroniche collegate tra loro devono rispettare per poter comunicare.

Esistono un gran numero di protocolli di rete, operanti ai diversi livelli della pila ISO/OSI (o TCP/IP), in base al tipo di servizio o comunicazione che devono garantire.

È necessario non confondere la definizione del protocollo con le sue implementazioni; il protocollo deve descrivere in modo non ambiguo le regole e le modalità di comunicazione, tralasciando, ove possibile, quelli che rappresentano i dettagli realizzativi e strumentali, mantenendosi ad un elevato livello di astrazione. Un design di protocollo troppo rivolto verso la singola implementazione porta inevitabilmente a limitarne le possibilità ed i futuri sviluppi da parte di sviluppatori terzi. Ai fini della massima adozione, il protocollo deve essere chiaro, facilmente comprensibile e, laddove ve ne siano le condizioni, aperto. L'implementazione di un protocollo invece deve recepire le direttive del protocollo, senza violarne la descrizione; aggiungere ulteriori specifiche ad un protocollo condiviso ne fa conseguire una scarsa compatibilità con altre implementazioni del medesimo protocollo.

Tutte queste considerazioni sono state utilizzate nella definizione e nel design del protocollo MoM e della sua implementazione (libmom).

6.1 Perché un nuovo protocollo

Esiste un grandissimo numero di protocolli di comunicazione ideati per i singoli scenari di utilizzo presentati in questo lavoro, in particolare in ambito domotico ed automotive. Apparentemente potrebbe risultare saggio basarsi su uno dei protocolli esistenti, intengrandone l'implementazione all'interno del sistema, ma non è necessariamente la scelta migliore. In questo paragrafo saranno presentati i protocolli commercialmente più diffusi, valutandone la compatibilità con gli scopi prefissi.

OpenWebNet[6]: nato per astrarre il bus SCS¹ per sistemi di Home Automation, si basa sullo scambio di messaggi tra i vari elementi del sistema, individuando i gateway tra la rete ed il bus SCS.

PRO: uno standard largamente utilizzato. Prevede Gateway con gli standard KNX e DMX

CONTRO: è pensato per funzionare su reti ethernet, seriali e ZigBee, ma non WiFi. Non prevede l'uso per scambio di messaggi di testo, o di comandi complessi. L'implementazione principale del protocollo è proprietaria, quindi richiederebbe una riscrittura dell'implementazione per gli scopi domotici, ed un'integrazione per gli altri scenari. Non consente il controllo diretto dei Pin.

DMX[33]: Digital Multiplex è uno standard di comunicazione ideato per l'iluminazione di scena.

PRO: ideato per il controllo diretto di pin e dispositivi

CONTRO: non prevede scambio di messaggi e comandi complessi. Estremamente legato allo standard fisico di trasmissione, assolutamente incompatibile con lo standard wifi.

KNX[3]: standard di building automation, ideato per il controllo delle varie parti della casa, approvato come standard europeo (EN 50090 - EN 13321-1) e mondiale (ISO/IEC 14543).

PRO: è indipendente dalla piattaforma di utilizzo;

CONTRO: ideato per vari mezzi trasmissivi tra i quali twisted pair(4880/9600 b/s), PowerLine (110 e 132 KHz), Radio Frequency (868 MHz) ed Ethernet, ma non WiFi. L'uso di questo standard richiede il pagamento di alcune royalty.

ANT[23]: protocollo di comunicazione che si appoggia su rete wireless mesh per le operazioni Home Automation

PRO: in assoluto quello che si avvicina di più alle necessità progettuali.

CONTRO: tecnologia proprietaria, con royalty, ed utilizzo di una tecnologia ULP (Ultra low Power) Wireless specifica, non adatta ad un dispositivo general purpose.

La situazione non migliora nel settore Automotive, dove esistono un gran numero di protocolli di comunicazione definiti In-Vehicle, ovvero tra elementi presenti all'interno del veicolo (ad esempio il CAN-Bus), ma di cui è ancora in fase di studio la comunicazione “tra veicoli”.

Ad ogni modo c'è chi sta lavorando a sistemi automotive che sfruttino le potenzialità delle reti mesh: risultano in fase di sviluppo diversi sistemi con queste caratteristiche quali Uncrashable Cars della Carnegie Mellon University [37], la Crash Avoidance tech di Toyota, il Volkswagen Autopilot e per ultimo il sistema sviluppato da Ford, denominato Ford-Fi. Al momento attuale si è stati in grado di fornire documentazione su questi prodotti.

¹SCS, Acronimo di Sistema di Cablaggio Semplificato, sviluppato da BiTicino per la domotica.

In generale si può dire che questi settori dispongano indubbiamente di un elevato numero di protocolli validi se considerati nel loro specifico campo di azione, ma estremamente ottimizzati e specifici. Nella maggior parte dei casi, questi protocolli lavorano ad un livello di astrazione molto basso, avvicinandosi molte volte a specifiche del livello fisico di trasmissione. L'uso di dispositivi quali schede di rete wireless generiche in ambiente domotico è considerato a tutt'oggi poco vantaggioso se paragonato a tecnologie ULP Wireless o a standard quali Bluetooth e ZigBee.

Un'altra importante valutazione da fare su molti di questi protocolli e standard, è quella sull'eventuale licenza di utilizzo delle specifiche e del software (vedere sezione 4.1.1): pur disponendo delle specifiche liberamente (cosa non sempre possibile) non è detto che si possa implementare una propria versione senza incappare nel pagamento di diritti o brevetti.

Come si potrà vedere nella sezione successiva, si necessita di un protocollo compatto, semplificato ma svincolato dal livello fisico e dal livello IP; in questo caso tale servizio è offerto dalle schede WiFi (livello fisico), dalla mesh BatMan (livello Mac) ed il kernel Linux (Livello IP), spostando quindi l'attenzione sul livello Applicazione.

6.2 Il protocollo MoM

Il protocollo MoM si basa sullo scambio di messaggi tra dispositivi connessi ad una rete mesh. MoM opera a livello 4 dello stack TCP/IP (Applicazione), appoggiandosi, almeno in questa versione, al protocollo IP, trascurando, di fatto tutto ciò che vi sia al di sotto del livello 3.

La base del protocollo è la definizione del messaggio; in questa sezione si andrà ad analizzare la struttura del messaggio MoM e le potenzialità offerte.

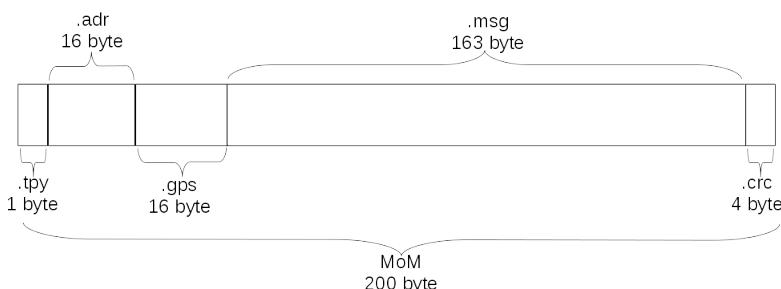


Figura 6.1: Struttura del messaggio MoM

Come visibile nella figura 6.1 il messaggio è a lunghezza fissa (200 byte) ed è composto dalle seguenti parti

Type (.tpy): campo ad 8 bit che definisce la tipologia del messaggio

Address (.adr): campo a 16 byte che definisce l'indirizzo del mittente (opzionale)

GPS (.gps): campo a 16 byte che contiene le coordinate geografiche del mittente (opzionale)

Message (.msg): campo da 163 byte che contiene messaggi di testo ed eventualmente comandi (opzionale)

CRC (.crc): campo da 4 byte che contiene un controllo di ridondanza ciclica del messaggio.

Come si sarà potuto notare tutti gli elementi ad eccezione del primo, sono considerabili opzionali; ciò vuol dire che in fase di elaborazione del messaggio questi campi possono essere non impostati (in accordo con quanto definito nel campo tipo), senza però intaccare la lunghezza complessiva del messaggio che deve rimanere costante.

6.2.1 Il campo Tipo

Il campo tipo (o dall'inglese “type”) è l'elemento fondamentale del messaggio, in quanto ne definisce il contenuto e la conseguente interpretazione. Composto da 1 byte (8 bit), descrive due differenti modalità operative, distinguendole sulla base del valore del bit più significativo (bit7); pin control (controllo pin) e message (messaggio).

pin control

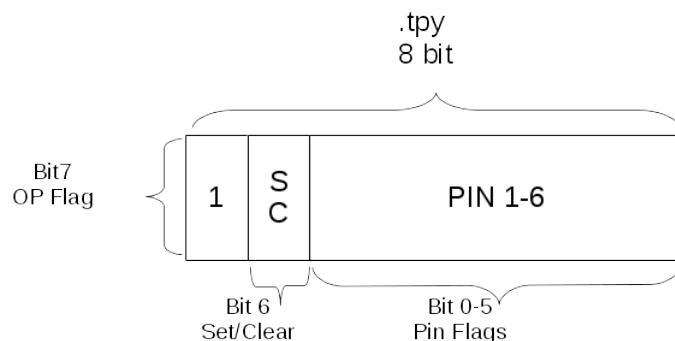


Figura 6.2: Campo tipo nella modalità pin control

Questa modalità è stata ideata per poter programmare un numero limitato (al massimo 6) di pin (ad esempio dei GPIO) in modo semplice ed efficace, come ad esempio in uno scenario domotico o automotive. Utilizzando solo il campo tipo, riduce notevolmente i bit da elaborare per dispositivi poco prestazionali o da ritrasmettere in caso di un gateway verso un altro layer fisico dalle velocità, come una porta seriale².

La struttura del campo tipo in questa modalità è la seguente:

OP flag (bit7): impostato ad 1 identifica la modalità operativa pin control.

Set/Clear flag (bit6): con valore 0 impone una operazione di Clear (il valore di uscita del pin viene portato a 0). Con il valore 1 impone una operazione di Set (il valore di uscita del pin viene portato a 1).

²Molti dispositivi domotici di nuova generazione utilizzano delle porte seriali verso degli specifici attuatori, o bus di comunicazione interni.

Pin flags (bit0-5): i pin su cui verrà effettuata l'operazione selezionata. Ogni singolo bit può essere 0 o 1, e quindi il valore del campo può oscillare tra 0 e 63.

La scelta di utilizzare un flag di Set/Clear è in realtà necessaria in considerazione del fatto che chi invia il messaggio potrebbe non essere a conoscenza dello stato attuale del sistema.

Utilizzando una configurazione numerica (ovvero passando direttamente il valore della configurazione dei pin), esiste il pericolo di cambiare il valore di un pin non contemplato nel messaggio inviato, e di cui il mittente non può conoscere la condizione attuale. Con la Set/Clear flag, utilizziamo il campo Pin solo per definire ad 1 i pin sui quali dovrà essere fatto un cambio di stato, ignorando completamente quelli impostati a 0.

Questa precauzione si traduce in un costo relativamente modesto, in quanto sono richiesti solamente due messaggi per dare una configurazione arbitraria a tutti i pin previsti.

message

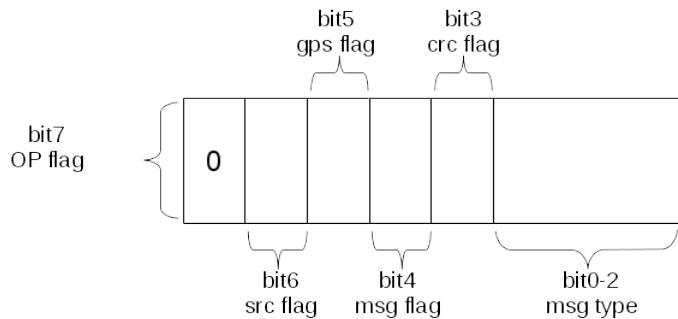


Figura 6.3: Campo tipo nella modalità message

La modalità message è quella progettata per l'invio e la ricezione di messaggi attraverso i vari dispositivi collegati alla medesima rete. In questo caso, la struttura dei bit di questo campo offre una descrizione di come sarà composto il messaggio, delle parti da elaborare e dalla tipologia di messaggio.

La struttura del campo tipo in questa modalità è la seguente:

OP Flag (bit7): impostato a zero identifica la modalità “message”.

src flag (bit6): impostato ad 1 indica che il campo “address” è stato impostato, a 0 altrimenti.

gps flag (bit5): impostato ad 1 indica che il campo “GPS” è stato impostato, a 0 altrimenti.

msg flag (bit4): impostato ad 1 indica che il campo “Message” è stato impostato, a 0 altrimenti.

crc flag (bit3): impostato ad 1 indica che il campo “crc” è stato impostato, a 0 altrimenti.

msg type (bit0-bit2): in base al valore identifica il tipo di messaggio.

Lo scopo delle “flag” (bit3-6) è quello di rendere di immediata comprensione per il ricevente il contenuto del messaggio, indicando cosa guardare e cosa ignorare. Il campo “msg type” offre maggiori informazioni sul tipo di messaggio inviato, e su come trattarlo sulla base delle configurazione binaria:

NUL_MSG (0 0 0) Messaggio Nullo: indica un messaggio vuoto (utilizzato per motivi di test)

TXT_MSG (0 0 1) Messaggio di testo: il messaggio contiene del testo

RSV_MSG (0 1 0) Messaggio riservato: il messaggio contiene un testo protetto

CMD_MSG (0 1 1) Comando: il corpo del messaggio contiene un codice comando da eseguire.

INF_MSG (1 0 0) Informazioni: il testo del messaggio contiene informazioni generiche.

HLP_MSG (1 0 1) Richiesta di aiuto: con o senza testo, può essere utilizzato come richiesta di aiuto.

WRN_MSG (1 1 0) Avviso: con o senza testo può segnalare un imminente pericolo.

EME_MSG (1 1 1) Emergenza: con o senza testo, può essere utilizzato per segnalare una situazione di estrema gravità.

Il “msg type” di tipo CMD_MSG permette di specificare una serie di comandi “user defined” (definiti dall’utente o dall’implementazione del protocollo) differenti da quelli visti nella modalità “pin control”. Tali comandi possono essere utilizzati per specificare istruzione complesse che necessitano di parametri.

Il TXT_MSG può essere utilizzato per comunicare semplici messaggi di testo attraverso altri dispositivi connessi con il protocollo MoM. Analogamente RSV_MSG può essere utilizzato per specificare che il testo è soggetto ad un qualche tipo di protezione (es. crittografia).

Le tipologie di messaggio WRN_MSG e INF_MSG sono intese per essere utilizzate da eventuali broadcaster (ovvero soggetti che inviano a tutta la rete su una vasta area) per inviare informazioni di carattere generale o avviso quali traffico, condizioni meteo, allerte, avviso di pericoli, interruzioni di servizio o notizie in generale.

I messaggi di tipo HLP_MSG e EME_MSG sono pensati per essere utilizzati dai singoli utenti per specificare una necessità di intervento medico sanitario, una situazione di pericolo per la sicurezza personale o una grave emergenza (attentati, incendi, situazioni di pericolo pubblico).

6.2.2 Il campo Message

Il campo Message (o messaggio) è stato ideato per contenere diversi tipi di messaggio di testo al suo interno ed è composto da due parti:

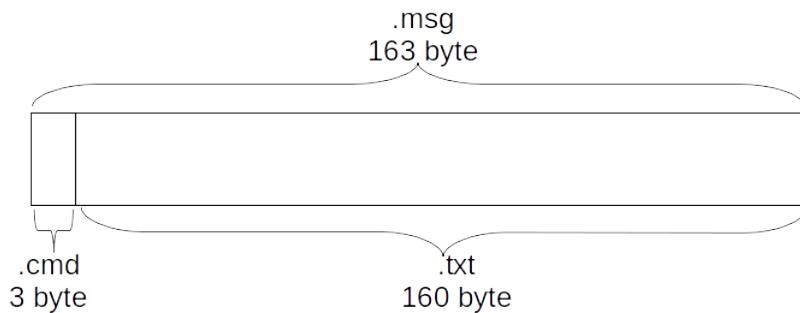


Figura 6.4: Campo messaggio

command (.cmd) 3 Byte: il campo comando, dedicato alle descrizioni di comandi "user defined".

text (160 Byte): contenente il testo del messaggio, o parametri opzionali al comando specificato.

In base a quanto definito all'interno del campo tipo, può essere utilizzato in due differenti modalità: testo e comandi.

Modalità testo

Questa modalità è quella predefinita, ed è quella abilitata da tutti i "msg type" ad eccezione del CMD_MSG. In questa modalità, il campo comando è riempito con una stringa 3 caratteri ASCII³ rappresentanti del "msg type" (NUL, TXT, RSV, INF, HLP, WRN, EME) in accordo con quanto predefinito nel campo tipo del messaggio. Il campo testo è predisposto per essere di 160 caratteri ASCII.

Modalità Comandi

Questa modalità viene abilitata dal "msg type" CMD_MSG all'interno del campo tipo. In questo caso, il campo testo non contiene testo ma gli eventuali parametri in ingresso dei comandi chiamati. Tale modalità è stata lasciata aperta alla definizione della implementazione.

L'idea alla base di questa modalità è di consentire agli sviluppatori di implementazioni software del protocollo di aggiungere comandi specifici per la piattaforma o per l'implementazione stessa.

Un potenziale ambito applicativo potrebbe essere la possibilità di lanciare comandi a basso livello sul sistema (ad esempio sui bus lenti quali I2C, I2S, CAN, SPI, seriali o definire altri GPIO da utilizzare), per scenari domotici o automotive.

La definizione del comando avviene nel campo command all'interno del campo testo, utilizzando 3 caratteri ASCII arbitrari, ad esclusione di quelli utilizzati per la modalità testo (NUL, TXT, RSV, INF, HLP, WRN, EME). Un possibile esempio di questa modalità potrebbe essere la trasmissione di audio digitale ad un dispositivo dotato di bus I2S: considerata la modesta velocità del bus

³ASCII è l'acronimo per American Standard Code for Information Interchange, ovvero un modello di codifica numerica per i caratteri di testo.

(solitamente intorno ai 48KHz) e la portata del bus (32bit), sarebbe possibile incapsulare 5 pacchetti I2S in un singolo messaggio MoM. Il carico richiesto sarebbe di circa 9600 pacchetti MoM al secondo, corrispondente ad una richiesta di banda di circa 1920Kb/s (molto al di sotto del potenziale di una rete WiFi odierna).

6.3 Implementazione software di MoM

Dopo aver creato il protocollo, mostriamo un’implementazione software del protocollo MoM realizzata espressamente per questo progetto. Avendo definito un protocollo piuttosto ampio dal punto di vista delle possibilità, si tratterà di un’implementazione parziale, ma in grado di testare gli aspetti significativi del protocollo. In questa versione, ci saranno tre componenti separati:

- libmom (libmom.so)
- mom_client (momcl)
- mom_server (momsvr)

6.3.1 libmom

Il componente libmom, come suggerito anche dal nome è rilasciato sotto forma di libreria dinamica (o libreria condivisa)⁴. La scelta di un simile approccio, che non è sempre la predefinita in ambienti embedded (dove talvolta si preferisce utilizzare librerie statiche), è dovuta alla necessità di poter utilizzare contemporaneamente più eseguibili che sfruttano il protocollo evitando di ricaricare il codice per ogni istanza.

Una libreria condivisa, per essere sfruttata correttamente dagli sviluppatori delle applicazioni, deve offrire a sua volta due elementi distinti e separati:

Il file binario della libreria: il binario deve essere disponibile sia in fase di linking che durante l’esecuzione del programma.

Gli header della libreria: sono file di testo, contenenti codice. Nello specifico sono presenti le strutture dati, le definizioni ed i prototipi delle funzioni.

Header della libreria

La libreria “esporta”⁵ i seguenti header file

sys_gps.h: esporta le strutture dati, le definizioni ed i prototipi di funzione relativi all’acquisizione delle informazioni GPS. Il prefisso “sys” sta ad indicare che verrà utilizzata un’interfaccia di sistema per l’accesso al dispositivo GPS (gpsd).

⁴Le librerie separano il codice in più elementi separati. In particolare una libreria dinamica consente il caricamento del codice della libreria solo se necessario. Tale “dynamismo” risulta particolarmente efficiente in caso di sistemi complessi con esecuzione non contemporanea di più librerie. Sono presenti sia nei sistemi Microsoft (.DLL) che nei sistemi Unix/Linux (.so). L’utilizzo di una libreria condivisa richiede particolare attenzione in fase di compilazione della stessa, e durante la fase di “linking” dei binari che sfruttano la libreria.

⁵Termine gergale atto ad indicare gli elementi messi a disposizione.

sys_gpio.h: esporta le strutture dati, le definizioni ed i prototipi di funzione relativi al controllo, scrittura e lettura dei GPIO. Il prefisso “sys” sta ad indicare che verrà utilizzata un’interfaccia di sistema per l’accesso ai GPIO (/sys/class/gpio).

mconnect.h: esporta le strutture dati, le definizioni ed i prototipi di funzione relativi alla connessione con altri dispositivi attraverso la rete.

mmessage.h: esporta le strutture dati, le definizioni ed i prototipi di funzione relativi alla composizione e alla interpretazione dei messaggi MoM. Questo header file è il fulcro del protocollo MoM in quanto contiene la definizione della struttura dati del messaggio e come esso venga manipolato.

I metodi della libreria

In questa sezione verranno presentate alcune funzioni esempio di questa implementazione⁶.

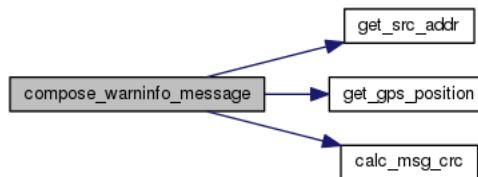


Figura 6.5: Grafico delle chiamate di `compose_warninfo_message()`

- Funzioni di messaggio:

compose_set_pin_message(): Compone un messaggio per la scrittura diretta dei pin eseguendo un’operazione di “set”. Accetta in ingresso un elemento di tipo “mesg” e la configurazione dei pin sotto forma di numero intero ad 8bit. Analoga funzione, ma per le operazioni di clear è **compose_clear_pin_message**, che utilizza i medesimi parametri.

compose_text_message(): Compone un messaggio di testo semplice. Prende in ingresso un elemento di tipo mesg, ed il testo da inserire nel corpo del messaggio.

compose_warninfo_message(): Compone un messaggio informativo o di avviso. Oltre alla struttura dati mesg e il testo, accetta il parametro msg_type che ne descrive il tipo.

- Funzione di controllo GPIO

reserve_gpio(): Funzione per l’inizializzazione di un GPIO tramite sysfs. Ogni GPIO deve essere inizializzato ed impostato nella corretta direzione (**set_direction**) per poter essere utilizzato dalla libreria.

write_gpio(): Scrive il valore (0 o 1) all’interno del GPIO. Analogamente la **read_gpio** ne esegue la lettura.

⁶Per motivi tipografici le funzioni verranno mostrate prive degli argomenti e dei valori di ritorno. Tale informazioni sono reperibili all’interno degli header file.

int led_pulse(): Esegue l'accensione e lo spegnimento di un led ad intervalli di tempo regolari per un determinato periodo tramite le scritture su GPIO

- Funzioni di connessione:

mconnect_send(): Invia un messaggio MoM ad un determinato indirizzo IP (implementazione al momento limitata su IPv4).

find_neighborhood(): Cerca gli indirizzi dei dispositivi prossimi aggiungendoli ad una lista di “vicini”. Il numero massimo dei vicini è definito nell’implementazione.

neighbor_send(): Invia uno specifico messaggio a tutta la lista dei vicini.

Per brevità sono state descritte solamente alcune delle funzioni esportate dalla libreria, cercando di mostrare quelle più significative.

6.3.2 MoM client

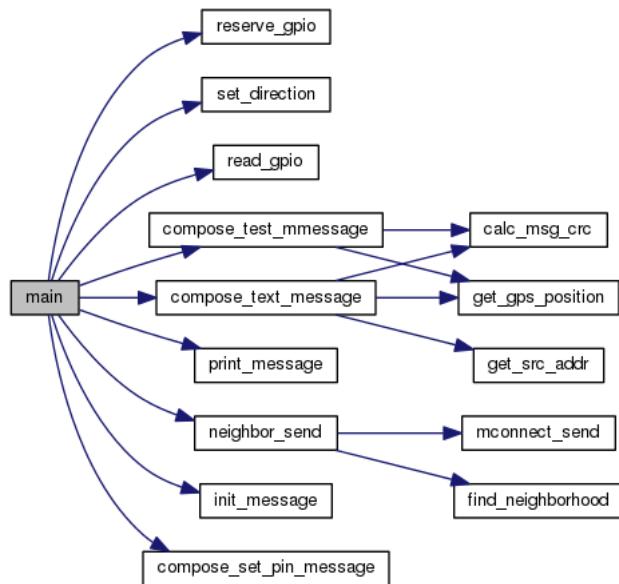


Figura 6.6: Grafico delle chiamate di momcl

Come è facilmente comprensibile dalla figura 6.6, la parte client realizzata in questo programma esempio, richiama buona parte delle funzioni di libreria di libmom, per comporre, inviare messaggi e per operare sull’attuatore.

6.3.3 MoM server

Differentemente dalla parte client, la parte server, è (volutamente) svincolata dalle funzioni della libreria, se non per quanto riguarda la parte relativa agli attuatori. Nella figura 6.7 è visibile come, alla ricezione di un messaggio, venga eseguita la segnalazione luminosa tramite led.

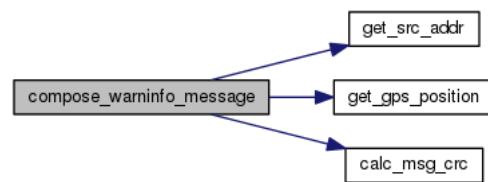


Figura 6.7: Grafico delle chiamate di momsrv

Alexjan Carraturo

Capitolo 7

Scenari di utilizzo

In questa sezione verranno presentati alcuni possibili ambiti applicativi del progetto con le relative specifiche.

7.1 Automotive

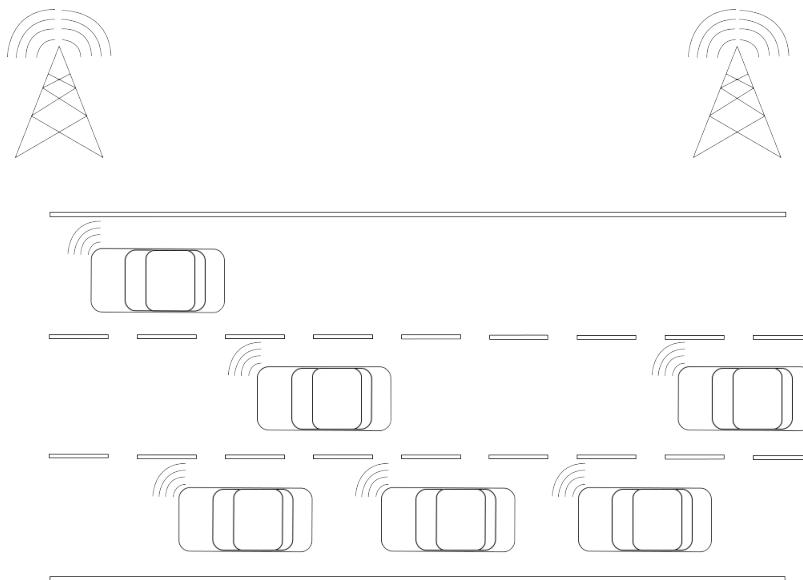


Figura 7.1: Caso d'uso Automotive

Scenario: Stradale o autostradale

Elementi:

- Automobili dotate di dispositivo mesh
- Antenne lungo il percorso (opzionale)

Descrizione scenario:

Presupponendo che le auto dispongano del dispositivo ideato, dotato di pulsante e luce, e posto che siano collegate tutte alla stessa rete mesh è possibile comunicare con le automobili vicine pur non conoscendone i conducenti.

Automobili che viaggiano a breve distanza e a velocità simili possono “riconoscersi” nella rete mesh, attraverso la tabella degli “originators”; periodicamente, al variare delle automobili prossime al veicolo in esame, la tabella viene aggiornata, avendo una cognizione del numero di automobili vicine (limitato dalla portata del WiFi), pur non conoscendone l’identità.

In caso di pericolo il veicolo può premere il bottone di segnalazione (ipoteticamente associato al tasto di emergenza delle auto) inviando un messaggio attraverso il protocollo MoM. Le altri automobili presenti considerate prossime (verosimilmente leggendo la lista degli “originators”) ricevono il messaggio di emergenza e, tramite l’attivazione dell’indicatore luminoso informano il conducente dell’eventuale situazione di pericolo, in modo che possa prendere opportuni provvedimenti. Il messaggio MoM contiene anche le informazioni GPS relative al veicolo che ha lanciato il messaggio. Applicando un’operazione di filtraggio è possibile limitare la propagazione del messaggio oltre una certa distanza. Aumentando la pulsantiera a disposizione sarebbe possibile aumentare il tipo di segnalazione possibile, sfruttando il campo “type” del protocollo MoM; un esempio pratico potrebbe essere la segnalazione agli automobilisti prossimi di strumenti per il controllo elettronico della velocità.

Descrizione scenario opzionale:

Aggiungendo delle antenne lungo il tragitto, è possibile mantenere la rete mesh anche con una bassa intensità di traffico. Le antenne, collegate ad una stazione centrale (anche in un solo punto), potrebbero propagare nella rete messaggi MoM, utilizzando il campo testo per delle informazioni utili, ed il campo GPS per localizzarle. Con una opportuna organizzazione della rete sarebbe possibile ritrasmettere a distanze programmate informazioni su traffico, incidenti, blocchi stradali ed altro. Tale sistema potrebbe inoltre raccogliere informazioni dalle stesse segnalazioni degli utenti o, in modo anonimo, elaborare delle statistiche sul traffico.

In caso di guasto sarebbe inoltre possibile, attraverso il campo messaggio e con un opportuno collegamento alla centralina della automobile (non previsto in questo progetto), specificare il tipo di guasto nel messaggio MoM.

Note implementative: Sulla base di questo scenario sono stati pensati i “Warning Message” e gli “Info Message” definiti nel protocollo MoM (sezione 6.2.1) e l’utilizzo di un campo specifico per la trasmissione della posizione tramite coordinate GPS. Tale scenario ha ispirato tutte le scelte relative alla sicurezza del dispositivo e alla non persistenza dei dati salvati (sezione 5.5.1).

Possibilità implementative future: Lo scenario “automotive” permette di ignorare o perlomeno trascurare il problema del consumo energetico, consentendo quindi l’utilizzo di dispositivi WiFi dotati di maggior potenza, allargando il raggio di azione.

Limiti: La maggior variabilità dal punto di vista della composizione della maglia, impone di aumentare la frequenza di aggiornamento della tabella degli originators. Il settore automotive ha degli stretti parametri di validazione e verifica per i prodotti legati alla sicurezza. Il SoC scelto non dispone di CAN¹

¹Controlled Area Network, standard bus ideato per la comunicazione dei dispositivi all’interno dei veicoli.

7.2 Domotica

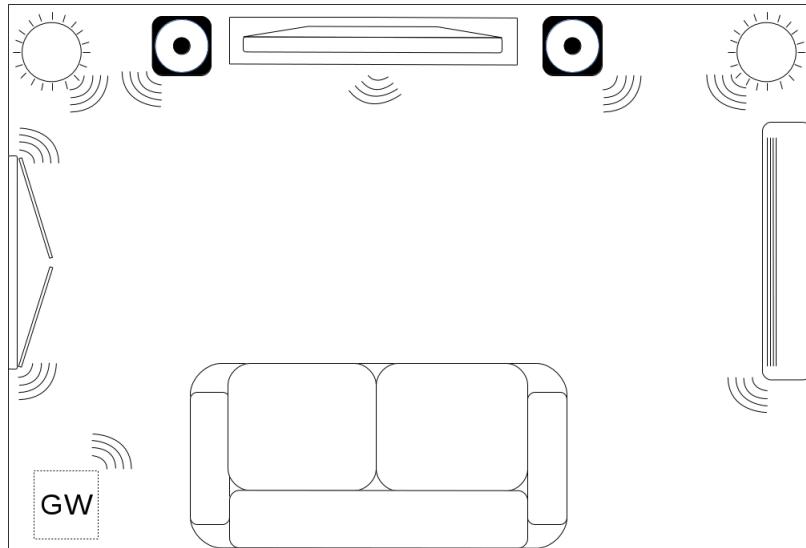


Figura 7.2: Caso d'uso domotico

Scenario: Domotico (Home Automation)

Elementi:

- Elettrodomestici con il dispositivo mesh
 - Televisione
 - Stereo
 - Serranda elettrica
 - Condizionatore/Pompa di calore
 - Luci
- Dispositivo di controllo (PC/Smartphone)
- Gateway (opzionale)

Descrizione scenario:

Gli elettrodomestici di vario genere o sorta dispongono del dispositivo mesh, e sono collegati alla stessa rete. Ricevono messaggi MoM che possono segnalare l'eventuale semplice accensione e spegnimento dell'elettrodomestico (es luci). Utilizzando il campo testo del messaggio MoM è possibile specificare comandi più articolati, quali la temperatura da mantenere nella stanza attraverso il condizionatore/pompa di calore, l'altezza della serranda, il volume e la canzone da ascoltare nello stereo, o il canale da vedere sulla televisione. Un simile schema sarebbe implementabile anche per altri elementi della casa (lavatrice, lavastoviglie, riscaldamento, luci).

A differenza dello scenario Automotiv, quello Domotico richiede grande attenzione al consumo del dispositivo mesh; essendo progettato per essere un dispositivo mobile, l'impatto sul consumo in fase di "stand-by" sarebbe limitato,

inoltre, la scarsa frequenza di spostamento e la vicinanza dei nodi, consentono di abbassare la frequenza di aggiornamento e di utilizzare dispositivi WiFi a basso consumo (e a portata ridotta).

In sostanza si tratta di un ribaltamento del concetto alla base del celebrato IoT (Internet of Things); i dispositivi sono collegati tra di loro, e sono controllabili da un elemento collegato alla stessa rete (dispositivo di controllo), senza essere collegati ad Internet.

Inoltre un sistema così concepito non richiede alcun tipo di cablaggio aggiuntivo per gli elettrodomestici, né l'utilizzo di standard wireless proprietari (es. ZigBee). A differenza di un modello centralizzato, il fallimento di un singolo dispositivo non influisce nella comunicazione tra gli altri.

Descrizione scenario opzionale:

Con l'aggiunta di un gateway tra la rete mesh e quella internet, i dispositivi potrebbero essere collegati alla rete internet pur non essendo prossimi a sufficienza al gateway. In una casa particolarmente grande, il dispositivo più remoto può utilizzare i nodi intermedi per raggiungere il gateway ed essere collegato.

Note implementative: Sulla base di questo scenario è stata ideata e creata la modalità “pin control” (sezione 6.2.1) del protocollo MoM. Sempre in considerazione di questo scenario sono state fatte le valutazioni relative al costo finale (sezione 5.1.3).

Possibilità implementative: Creazione di un dispositivo specifico per il controllo della casa, sulla base della implementazione vista in precedenza, ma specifico per il controllo diretto (es. sistema con interfaccia utente touch screen).

Svantaggi: L'integrazione con i diversi produttori è un processo complicato e non necessariamente compatibile con il modello di comunicazione creato.

7.3 Sicurezza personale

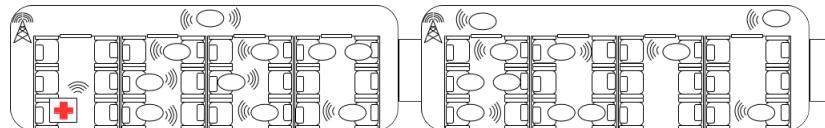


Figura 7.3: Gestione dell'emergenza

Scenario: Ambienti pubblici

- Metropolitane
- Treni
- Stazioni
- Navi
- Strutture sportive (Stadi e palazzi dello sport).

Elementi

- Persone con dispositivo mesh
- Personale di sicurezza con dispositivo mesh

- Gateway (opzionale)

Descrizione scenario:

In un ambiente pubblico, affollato o meno, può essere difficile segnalare un'emergenza, anche alle persone prossime. L'utente, utilizzando il bottone di emergenza del dispositivo mesh può segnalare il bisogno di assistenza o aiuto alle persone prossime, e qualora presenti, anche richiedere l'intervento di personale di sicurezza o medico/sanitario.

Descrizione scenario opzionale: L'utilizzo di antenne Gateway, oltre che potenziare la rete e renderla più affidabile, consentirebbe una migliore e più facile individuazione di coloro che richiedono assistenza.

Note implementative: Sulla base di questo scenario di utilizzo sono state fatte quasi tutte le considerazioni relative al risparmio energetico (sezione 5.1.4) e sulle dimensioni del prodotto finale (sezione 5.1.1). Inoltre questo scenario è quello che ha ispirato l'utilizzo delle immagini precaricate per il sistema (sezione 5.5.1 e l'idea di utilizzare MoM per messaggi di testo tra gli utenti (sezione 6.2.2

Possibilità implementative future: Sarebbe possibile, utilizzando il campo testo del messaggio MoM, inviare messaggi di testo a utenti prossimi ma sconosciuti. Le notevoli implicazioni dal punto "Social Networking" localizzato sarebbero ad oggi tutte da valutare, ma consentirebbero un cambio di paradigma nella comunicazione tra i singoli, con ripercussioni anche sul marketing e la profilazione.

Limiti: Non sono stati ancora fatti studi approfonditi sulla validità e la stabilità di una rete mesh basata su WiFi a velocità elevate.

7.4 Possibili sviluppi futuri

L'estrema versatilità dei sistemi embedded e delle reti wireless apre numerosi possibili scenari di utilizzo futuri e conseguenti possibilità di sviluppo. Quello che al momento sembra lo sviluppo più probabile ed interessante in questo settore è l'integrazione delle reti Mesh all'interno dei dispositivi mobili quali telefoni e tablet. Allo stato attuale dei sistemi mobili, il sistema che sembra più promettente dal punto di vista dell'integrazione sembrerebbe il sistema Android, in parte derivato da Linux: come già evidenziato nella sezione 4.3.1, Android ha alcuni limiti a livello di librerie di sistema che rendono questo lavoro di porting assolutamente non banale, richiedendo dei pesanti, ma comunque possibili, interventi a livello di sistema. Altri sistemi operativi mobile derivati da Linux sono ancora in fase di forte sviluppo, rendendo difficile un eventuale termine di paragone. I più promettenti da questo punto di vista sono Ubuntu touch e Tizen, che comunque non hanno assolutamente la base di utenti necessaria ad uno sviluppo su larga scala.

Un altro importante ambito di sviluppo nelle reti mesh è lo streaming di contenuti multimediali di vario genere: se da un lato le reti mesh aprono delle nuove possibilità, il carico trasmissivo di questa tipologia di traffico sembrerebbe essere incompatibile con il degrado delle prestazioni rispetto ad altre modalità di accesso. BATMAN offre, se pur in via sperimentale, il supporto a reti multicast, in grado di ottimizzare il traffico dati in una trasmissione "uno a molti".

Alexjan Carraturo

Appendice A

Ambiente di test

Durante lo sviluppo del progetto non è possibile lavorare direttamente su quella che è l’immagine finale di sistema. I numerosi tentativi dovuti alla realizzazione di una configurazione di kernel ottimale, o di verifica funzionamento del rootfs, rendono poco pratico il lavorare direttamente su un supporto fisico.

Un’altra significativa necessità, è legata alle fasi di sviluppo dell’implementazione software del protocollo MoM: per quanto sia possibile ad oggi eseguire un discreto lavoro con il “debug in place” (debug sul posto), nelle fasi iniziali può risultare essere assai più conviene provare il software in un ambiente simulato, con degli opportuni accorgimenti all’interno del codice. Per questo motivi, risulta decisamente utile creare un ambiente di test, differente da quello che sarà il risultato finale, ma comunque in grado di simularne il comportamento.

A.1 Alternative per lo storage

Non si può dire che esistano delle linee guida generali per tutti i dispositivi embedded che, come visto nella sezione dedicata all’hardware, sono suscettibili a numerose variazioni dal punto di vista della dotazione. Scendendo nello specifico delle evaluation board per SoC di fascia alta, la situazione tende ad essere più standardizzata; nella maggior parte dei casi (TI, ST, Freescale, Nvidia, RockChip, Mediatek per citarne alcuni) questi dispositivi dispongono di una presa di rete ethernet. Contrariamente a quanto avviene nel comune mondo desktop, in realtà, non è possibile definirla propriamente come un’unica scheda di rete, in quanto i livelli di questo collegamento possono (e sovente lo sono) essere disposti in “luoghi” differenti; il livello Mac viene ospitato all’interno del SoC, mentre il livello fisico (in gergo il “fisico” o phy) è collocato esternamente in prossimità (o internamente) della presa.

Da notare che, come avviene nel nostro specifico caso, in molti prodotti finali, sia il “fisico” che la relativa presa verranno rimossi, mentre rimarrà il livello MaC nel SoC, disabilitato a livello software.

L’aggiunta di questo componente nella evaluation board, ad esclusione di casi eccezionali (es. prototipazione hardware di rete), è da intendersi proprio a scopo di sviluppo e testing.

Grazie alle funzioni avanzate dei bootloader (vedere sezione 4.4) è infatti possibile caricare tutti gli elementi fondamentali del sistema attraverso la rete.

Per fare questo l'ambiente di test deve disporre di:

- Un PC Linux (possibilmente il sistema Host utilizzato come ambiente di sviluppo)
 - Un server NFS per la condivisione del rootfs
 - Un server TFTP per il caricamento delle immagini di Kernel e device-tree
- Collegamento ethernet tra Host e board (meglio se attraverso uno switch)
- Una porta seriale/cavo seriale.

Il termine porta seriale, utilizzato in questo contesto per chiarezza espositiva, in realtà potrebbe non essere il più indicato. Pur non rappresentando la totalità dei casi, nella maggioranza dei casi è possibile che la porta seriale non sia la classica RS-232 a 9 pin, ma una seriale TTL incompleta (termine utilizzato per indicare che solo alcuni fili sono realmente connessi) a voltaggio variabile (solitamente 3.3V o 5V). L'uso diretto di una seriale classica (con livelli molto più alti) potrebbe rivelarsi inutile e dannoso per la board stessa. Nelle board più moderne e di fascia alta, la funzionalità seriale è offerta tramite porte USB OTG collegate ad un apposito convertitore (usb2serial) già impostato al corretto voltaggio.

Nel nostro caso è stato utilizzato il convertitore USB2Serial TTL presentato nella sezione Hardware. Quanto al PC Linux, si fa riferimento a quanto già segnalato nella fase di configurazione dell'ambiente di sviluppo ma con due aggiunte: i server per nfs e tftp.

A.1.1 NFS

Come già visto nella sezione 5.5, NFS consente di esportare filesystem o parti di esse attraverso la rete. A differenza di molti altri server di simile funzionalità, questo è stato integrato all'interno del kernel Linux, limitando molto gli aspetti d'installazione a livello userspace. Nella configurazione Workstation di Fedora 22 non è necessario installare nulla. Gli unici accorgimenti sono quelli in fase di configurazione.

Nel file “/etc(exports”

```
/home/build/rootfs/bbb_mesh *(rw,async,no_root_squash)
```

Il primo path indica il luogo dove è presente in forma decompressa il rootfs che si vuole utilizzare. Il valore “*” sta ad indicare le restrizioni di accesso a livello IP (potrebbe essere sostituito dall'indirizzo di una rete o di un singolo host). I parametri tra parentesi rappresentano le opzioni di mount del rootfs. Non resta che abilitare ed avviare il servizio, attraverso le apposite utility di systemd

```
# exportfs -va
# systemctl enable nfs-config.service
# systemctl enable nfs-idmapd.service
# systemctl enable nfs-mountd.service
# systemctl enable nfs-server.service
# systemctl start nfs-config.service
# systemctl start nfs-idmapd.service
# systemctl start nfs-mountd.service
# systemctl start nfs-server.service
```

Alcuni accorgimenti in ambito sicurezza potrebbero essere necessari; in sistemi dotati di firewall infatti la richiesta NFS della board potrebbe essere rifiutata, in particolare per problemi relativi alle RPC (remote procedure call) del server NFS che rispondono su porta arbitraria (a meno di configurazione) alle richieste. In alcuni casi potrebbe essere necessario (anche se poco raccomandabile) disabilitare il firewall di rete.

A.1.2 TFTP

Come già visto nella sezione 5.5, TFTP consente di caricare file attraverso un protocollo FTP semplificato (ideale per ambienti semplici come i bootloader). Per l'installazione è necessario eseguire:

```
# dnf install tftp-server
```

Una volta operato è necessario operare sul file di configurazione /etc/xinetd.d/tftp

```
service tftp
{
    socket_type      = dgram
    protocol         = udp
    wait             = yes
    user             = root
    server           = /usr/sbin/in.tftpd
    server_args      = -s /home/build/tftpboot
    disable          = no
    per_source        = 11
    cps              = 100 2
    flags            = IPv4
}
```

Il percorso di “tftpboot” è una cartella non presente nel sistema, che deve essere creata e configurata manualmente. All'interno di questa cartella dovranno essere copiate le immagini di kernel e device-tree.

Anche in questo caso sono rischiesti alcuni accorgimenti sulla sicurezza: oltre ad aprire sul firewall (qualora non sia abilitata) la porta 69 UDP, è necessaria una modifica nel comportamento di sicurezza del sistema host. Molti sistemi recenti utilizzano un meccanismo noto come SELinux, abilitato come scelta predefinita. Tale meccanismo non è molto compatibile con l'accesso di tftp, rendendolo di fatto inutilizzabile. Non sapendo quale sia il contesto di lavoro del sistema host, potrebbe risultare pratico impedire a SELinux di interferire con il lavoro di tftp-server:

```
# setenforce 0
```

A.1.3 Utilizzo di NFS/TFT in U-Boot

Contrariamente a quanto visto in precedenza, è necessaria una differente configurazione delle variabili di ambiente di U-boot (mostrate nella sezione 5.6) per poter utilizzare il boot da rete. Data l'estrema variabilità del sistema, e considerando il fatto di dover utilizzare il bootloader già presente all'interno della board, è poco proponibile l'utilizzo di un file uEnv.txt: in tal senso, è più opportuno passare volta per volta questi parametri tramite la console seriale messa a disposizione di uboot.

```
setenv ipaddr 192.168.127.7
setenv serverip 192.168.127.1
setenv gatewayip 192.168.127.1
setenv netmask 255.255.255.0
setenv ipstring 'ip
    ↪ =192.168.127.7:192.168.127.1:192.168.127.1:255.255.255.0:
    ↪ meshsystem:eth0:::'
setenv meshnetargs 'setenv bootargs console=console=tty00,115200n8
    ↪ root=/dev/nfs rootwait nfsroot=192.168.127.1:/home/build/
    ↪ rootfs/bbb_mesh,nolock,v3 rw $ipstring'
setenv meshnetcmd 'run meshnetargs; tftp 0x80200000 uImage_bbbmesh;
    ↪ tftp 0x80F80000 am335x-boneblack.dtb; bootm 0x80200000 - 0
    ↪ x80F80000'
```

Da notare che in questa configurazione è stato impostato un IP statico, anche se di frequente si preferisce utilizzare un server dhcp sul PC host in grado di fornire tutte queste informazioni.

A.2 Ambiente test della rete mesh

Per poter testare il funzionamento e la configurazione della rete Mesh, così come per fare il debug dell'implementazione del protocollo MoM, è stato necessario un secondo computer che per comodità definiremo “test-*pc*”. Utilizzando un normale PC, dotato di schermo e tastiera è stato possibile modificare l'implementazione software per inviare messaggi arbitrari alla rete mesh, consentendo la verifica in tempo reale dei pacchetti MoM in transito. Un simile livello di debug non sarebbe stato difficilmente raggiungibile utilizzando un'altra board priva di schermo e tastiera. Utilizzare il PC host, per quanto possibile, avrebbe potuto falsare i risultati di test, essendo già collegato alla board target tramite rete ethernet, e disponendo di molti software di configurazione automatica potenzialmente fuorvianti.

Come test-*pc* è stato utilizzato un Acer Aspire One A110, un computer semplice, datato e computazionalmente poco performante; contrariamente a quanto si potrebbe pensare, le scarse prestazioni del test-*pc* sono un fattore estremamente positivo in quanto consentono di avvicinarci quanto più possibile al profilo prestazionale della board.

Sulla falsa riga della scelta hardware è stata fatta la configurazione di sistema del test-*pc*: è stata utilizzata una Slackware Linux current, con un'installazione minimale. Si è cercato, per quanto possibile, di utilizzare la stessa dotazione software prevista per la board, con alcune eccezioni dovute a necessità particolari. Per il test-*pc* è stato realizzato uno specifico kernel con i driver wireless e con il modulo B.A.T.M.A.N.

Appendice B

Comandi Batman

B.1 Esempio di configurazione manuale

A seguire un esempio di configurazione manuale di una rete mesh, tramite linea di comando:

```
ifconfig wlan0 mtu 1560
ifconfig wlan0 down
iwconfig wlan0 mode ad-hoc essid mymesh channel 1
ifconfig wlan0 up
batctl if add wlan0
ifconfig bat0 192.168.127.3 up
```

In un primo momento viene caricato il modulo kernel batman-adv. La prima chiamata al comando ifconfig modifica il valore del MTU (come già visto nella sezione 2.2.1) per il corretto funzionamento della rete mesh. A seguire, sempre tramite ifconfig viene disabilitata la scheda wireless. Tale passaggio è fondamentale, in quanto moltissime schede WiFi non accettano un cambio di modalità (da “Managed” ad “Ad Hoc”) in corsa, vanificando di fatto le impostazioni di connessione successive.

Dopo aver disabilitato la scheda è possibile cambiare la modalità con il comando “iwconfig”: in questo caso la rete mesh ha come ESSID il valore “mymesh” su canale 1. Una volta riabilitata la scheda wifi (linea 4), è possibile utilizzare il comando batctl (B.3) per aggiungere la scheda WiFi in utilizzo al controllo del modulo Batman. Infine viene configurato un indirizzo IP per la nuova interfaccia “bat0” creata dal moudulo Batman.

B.2 Un esempio di meshconf.txt

A seguire un esempio del file di configurazione meshconf.txt

```
#cnf
BATFLAG=yes
MTU=1560
ESSID=mymesh
CHANNEL=5
WLAN=wlan0
IP=192.168.127.10
```

```
NETMASK=255.255.255.0
GATEWAY=192.168.127.1

MOM_SERVER=true
MOM_CLIENT=true
MOMCL_PARAM=""
MOMSRV_PARAM=""
```

Spiegazione dei parametri

BATFLAG: indica se la rete utilizzerà il protocollo Batman

ESSID: specifica il valore ESSID della rete

CHANNEL: specifica il canale da utilizzare

WLAN: il nome del dispositivo WiFi da utilizzare per la rete mesh

IP: specifica se il dispositivo utilizza un indirizzo ip statico o dhcp. In caso di ip statico è possibile specificare anche i valori di netmask e gateway (opzionale)

MOM_SERVER: se uguale a “true” abilita all’avvio il servizio momsrv.

MOM_CLIENT: se uguale a “true” abilita all’avvio il servizio momcl

B.3 batctl

batctl[20] è il tool di configurazione e debug per batman-adv. Tutte le operazioni di configurazione di batman-adv sono fatte tramite *sysfs*¹ e batctl ha solo una funzione di interfacciamento, con il quale è possibile ad esempio aggiungere o rimuovere interfacce alla rete mesh, impostare o cambiare la configurazione corrente del modulo batman-adv e abilitare o disabilitare caratteristiche.

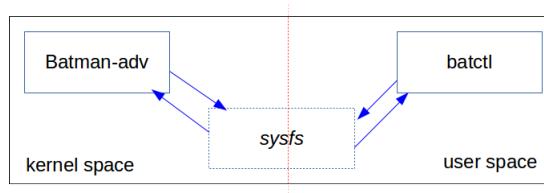


Figura B.1: Rappresentazione del funzionamento di batman-ctl tramite sysfs

Il comando batctl, per poter essere utilizzato necessita dei permessi da “superutente”. Una rapida panoramica sulle modalità di utilizzo[13].

interface|if: in assenza di parametri mostra la situazione attuale. Con add o del ed il nome del device di rete aggiunge o rimuove il dispositivo fisico alla lista di quelli attivi sul device virtuale.

orig_interval|it [intervallo msec]: in assenza di parametro mostra la situazione attuale. Come parametro accetta un numero intero di millisecondi per impostare l’intervallo di tempo degli “originator message”.

¹Sysfs[29] è un filesystem virtuale fornito dal kernel linux per esportare informazioni relative ai sottosistemi del kernel, all’hardware installato e ai device driver associati.

ap_isolation|ap [0|1]: utilizzato per abilitare e disabilitare la “ap_isolation”(Vedere sezione 2.2.1). Senza parametro mostra la situazione attuale.

bridge_loop_avoidance|bl [0|1]: abilita o disabilita la “bridge_loop_avoidance”(Vedere sezione 2.2.1). Questo supporto deve essere abilitato al momento della compilazione del modulo kernel. Senza parametro mostra la situazione attuale.

distributed_arp_table|dat [0|1]: abilita o disabilita la “distributed_arp_table”(Vedere sezione 2.2.1). Senza parametro mostra la situazione attuale.

aggregation|ag [0|1]: abilita o disabilita la ”OGM packet aggregation“. Senza parametro mostra la situazione attuale.

bonding|b [0|1]: abilita o disabilita il bonding (Vedere sezione 2.2.1). Senza parametro mostra la situazione attuale.

fragmentation|f [0|1]: abilita o disabilita la fragmentation (Vedere sezione 2.2.1). Senza parametro mostra la situazione attuale.

network coding [0|1]: abilita o disabilita il ”network coding“(Vedere sezione 2.2.1). Senza parametro mostra la situazione attuale.

multicast_mode|mm [0|1]: abilita o disabilita il ”multicast_mode“. Senza parametro mostra la situazione attuale.

loglevel|ll: imposta il livello di log. Va da un livello ”none“, praticamente privo di attività di log, a ”batman“ dove sono abilitati tutti i messaggi su routing, flooding e broadcasting

log: mostra i log di debug di batman-adv

gwmode|gw [off|client|server]: abilita o disabilita il ”gateway mode“, ed imposta il comportamento del nodo come client o come server.

routing_algo|ra: senza parametro mostra gli algoritmi di routing disponibili. Selezionando un algoritmo tra quelli disponibili a parametro, lo imposta per il routing.

statistics|s: recupera le statistiche dal modulo kernel badtman-adv.

ping|p: versione del comando ping² operativa a layer2, in grado di eseguire operazioni di ping anche con un bat-hostname o attraverso MAC address, oltre che con i classici IPv6 ed IPv4.

traceroute: versione del comando traceroute³ operativa a layer2 in grado di eseguire operazioni di traceroute anche con un bat-hostname o attraverso MAC address, oltre che con i classici IPv6 ed IPv4.

tcpdump|td: versione del comando ⁴ per batman.

²Ping è un programma che utilizza il protocollo ICMP per inviare una ECHO_REQUEST. Utilizzato solitamente per verificare se un host sia o meno attivo.

³Traceroute è un comando che utilizza il protocollo ICMP (TIME_EXCEEDED) per tracciare il cammino eseguito dai pacchetti dalla sorgente alla destinazione specificata.

⁴Tcpdump è un tool di analisi di rete per la visualizzare/salvare le informazioni relative ai pacchetti in transito.

B.4 Alfred

Alfred[11] (Almighty Lightweight Fact Remote Exchange Daemon) è un programma di tipo demone userspace per la distribuzione di informazioni arbitrarie su una rete mesh in modo decentralizzato, è stato ideato per la distribuzione di hostname, rubrica telefonica, informazioni dns e di amministrazione, previsioni metereologiche.

Tipicamente Alfred viene eseguito come demone in background ed accetta l'inserimento di informazioni attraverso il comando binario su terminale o dei programmi scritti appositamente per questo scopo per comunicare tramite unix socket. Una volta ricevuti i dati locali, il demone di Alfred si incarica di redistribuirle agli altri server con alfred.

Opzioni del comando alfred in modalità client.

- s |**–set-data** *data-type*: imposta nuovi dati da essere distribuiti per lo specifico "data-type", rappresentato da un valore intero a 8 bit (0 - 255). I valori da 0 a 63 sono riservati e non possono essere utilizzati attraverso il comando da terminale.
- r |**–request**: colleziona i dati dalla rete e li stampa sulla rete
- d |**–verbose**: mostra ulteriori informazioni in output.
- V |**–req-version**: specifica la versione dei dati usata dall'opzione "-s".
- M |**–modeswitch** *mode*: con il valore di "master" e "slave" modifica il comportamento del demone.
- I |**–change-interface** *interface*: cambia l'interfaccia utilizzata.

Opzioni del comando alfred in modalità server:

- i |**–interface** *interface*: specifica l'interfaccia sulla quale ascoltare
- b *batmanif*: specifica l'interfaccia batman (solitamente bat0) da utilizzare.
- m |**–master**: imposta il demone in master mode (accetta dati dai nodi "slave" e li mantiene sincronizzati con altri nodi master).
- c |**–update-command** *command*: specifica il comando da eseguire in caso di cambio dei dati.

Esiste una versione esclusivamente client di Alfred per android che prende il nome di Alfreda, ancora in via sperimentale.

B.4.1 Alfred-gpsd

Alfred-gps può essere utilizzato per distribuire le informazioni GPS dei singoli nodi all'interno di una rete mesh batman. Queste informazioni possono essere utilizzate per costruire una mappa geografica della rete mesh. Alfred-gpsd acquisisce le informazioni dal demone di sistema gpsd (o in alternativa può utilizzare dati forniti da line di comando).

Bibliografia

- [1] B.a.t.m.a.n. concept, 2011. <http://www.open-mesh.org/projects/open-mesh/wiki/BATMANConcept>.
- [2] ArduinoCC. Storia di arduino, 2015. <http://playground.arduino.cc/Italiano/StoriaDiArduino>.
- [3] Knx Association. Knx system specifications, 2013. <http://www.knx.org/media/docs/downloads/KNX-Standard/Architecture.pdf>.
- [4] Atmel. Atmega8, 2015. <http://www.atmel.com/devices/atmega8.aspx>.
- [5] Atmel. Sam3x8e, 2015. <http://www.atmel.com/devices/sam3x8e.aspx>.
- [6] BiTicino. Openwebnet language, 2000. http://www.myopen-legrandgroup.com/resources/own_protocol/default.aspx.
- [7] Buildroot. Buildroot, 2015. <http://buildroot.uclibc.org/about.html>.
- [8] David Johnson Corinna Aichele, Ntsibane Ntlatlapa. A simple pragmatic approach to mesh routing using batman. 2007.
- [9] Network Working Group. Nfs: Network file system protocol specification rfc1094, 1989. <https://tools.ietf.org/html/rfc1094>.
- [10] Network Working Group. The tftp protocol (revision 2) rfc1350, 1992. <https://tools.ietf.org/html/rfc1350>.
- [11] OpenMesh group. Alfred, 2015. <http://www.open-mesh.org/projects/alfred/wiki>.
- [12] OpenMesh group. Ap isolation, 2015. <http://www.open-mesh.org/projects/batman-adv/wiki/Ap-isolation>.
- [13] OpenMesh group. batctl manual, 2015. <http://downloads.open-mesh.org/batman/manpages/batctl.8.html>.
- [14] OpenMesh group. Batman-adv wiki, 2015. <http://www.open-mesh.org/projects/batman-adv/wiki/Wiki>.
- [15] OpenMesh group. Bridge loop avoidance, 2015. <http://www.open-mesh.org/projects/batman-adv/wiki/Bridge-loop-avoidance>.
- [16] OpenMesh group. Distributed arp table, 2015. <http://www.open-mesh.org/projects/batman-adv/wiki/DistributedArpTable>.

- [17] OpenMesh group. Fragmentation technical note, 2015. <http://www.open-mesh.org/projects/batman-adv/wiki/Fragmentation-technical>.
- [18] OpenMesh group. Multilink mesh, 2015. <http://www.open-mesh.org/projects/batman-adv/wiki/Multi-link-optimizations-technical>.
- [19] OpenMesh group. Network coding, 2015. <http://www.open-mesh.org/projects/batman-adv/wiki/NetworkCoding>.
- [20] OpenMesh group. Using batctl, 2015. <http://www.open-mesh.org/projects/batman-adv/wiki/Using-batctl>.
- [21] IEC. Iec 61131-3, 2013. <https://webstore.iec.ch/publication/4552>.
- [22] IETF. rfc 3626, 2008. <https://www.ietf.org/rfc/rfc3626.txt>.
- [23] Dynastream Innovations Inc. Ant message protocol and usage, 2007. <https://www.sparkfun.com/datasheets/Wireless/Nordic/ANT-UserGuide.pdf>.
- [24] Texas Instruments. am3358, 2011. <http://www.ti.com/lit/ds/symlink/am3358.pdf>.
- [25] Texas Instruments. Am335x power consumption, 2015. http://processors.wiki.ti.com/index.php/AM335x_Power_Consumption_Summary.
- [26] Texas Instruments. Am335x thermal consideration, 2015. http://processors.wiki.ti.com/index.php/AM335x_Thermal_Considerations.
- [27] ISO. Iso 13485:2003, 2003. http://www.iso.org/iso/catalogue_detail?csnumber=36786.
- [28] William Joy. An introduction to the c shell. 1886.
- [29] kernel.org. Sysfs documentation, 2015. <https://www.kernel.org/doc/Documentation/filesystems/sysfs.txt>.
- [30] Huawei Technologies LTD. Mesh technology white paper. 2011.
- [31] ST Microelectronics. 32f429idiscovery short description, 2015. <http://www.st.com/web/catalog/tools/FM116/SC959/SS1532/PF259090>.
- [32] Aruba Networks. Next-generation wireless mesh networks. 2011.
- [33] OpenDMX. Dmx512-a, 2015. <http://opendmx.net/index.php/DMX512-A>.
- [34] Yocto Project. Yocto project, 2011. <https://www.yoctoproject.org/about>.
- [35] Vincent Lenders Martin May Rainer Baumann, Simon Heimlicher. Routing packets into wireless mesh networks. 2007.

- [36] U-boot. U-boot image format, 2015. <http://www.denx.de/wiki/view/DULG/UBootImages>.
- [37] Carnegie Mellon Univeristy. Self driving car, 2013. http://www.cmu.edu/news/stories/archives/2013/september/sept4_selfdrivingcar.html.