

Sistemi embedded

esistono molte definizioni nessuna universalmente riconosciuta.
In generale con sistema embedded si intende **un dispositivo incapsulato** all'interno del sistema da controllare **progettato per una determinata applicazione** supportato da una piattaforma hardware su misura

caratteristiche di un sistema embedded

- sistemi privi di interazione umana
- capaci di resistere ad eventi dannosi, a vibrazioni e shock
- ripartire in modo autonomo
- possibilmente di dimensioni ridotte
- raggiungibile tramite un qualche tipo di connessione
- deve avere risorse necessarie per l'esecuzione dei processi indispensabili al suo funzionamento
- spesso con vincoli temporali (sistema real-time)

Cosa intendiamo noi per sistema embedded

è essenzialmente un sistema a microprocessore come un PC ma

- ★ dedicato a svolgere un particolare compito
- ★ ha risorse e dispositivi strettamente necessari
- ★ hardware dedicato
- ★ frequenze di CPU inferiori
- ★ basso consumo
- ★ poca memoria (anche di massa)

piattaforme per i sistemi embedded

- Una varietà molto più ampia di quella presente nel campo dei PC desktop
- processori Intel x86, Power PC, ARM, MIPS
- System on Chip, FPGA
- standard industriali come il PC104
- interfacce utente (display LCD, led..)

Sistemi operativi? No grazie

- ♦ un S.O richiede più risorse computazionali
- ♦ vincoli di specifica da rispettare
- ♦ sistemi non molto complessi che non necessitano di concorrenza
- ♦ gestione in tempo reale

Sistemi operativi? Sí grazie

- supporto per il multi-threading e multi-processing
- supporto protocolli di comunicazione, memorie di massa
- tempo di sviluppo minore
- librerie grafiche, tools di sviluppo, debugging
- porting verso altre piattaforme hardware

Sistemi proprietari o open-source ?

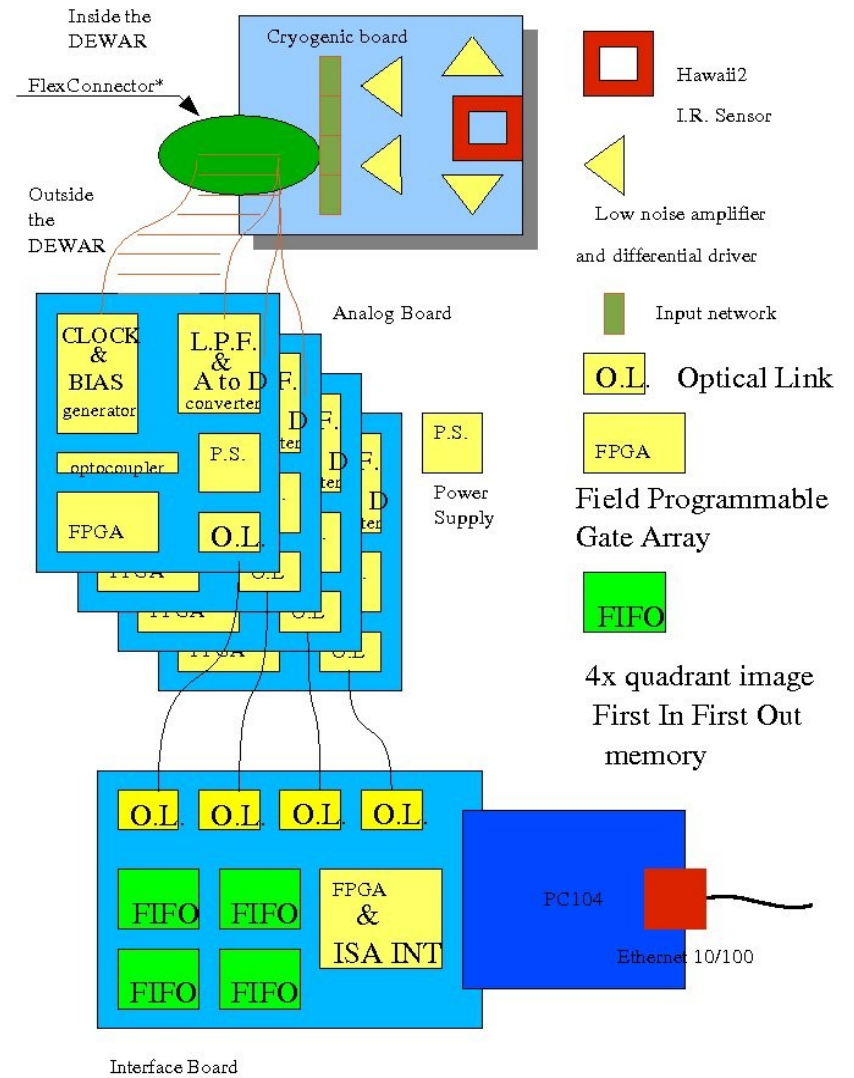
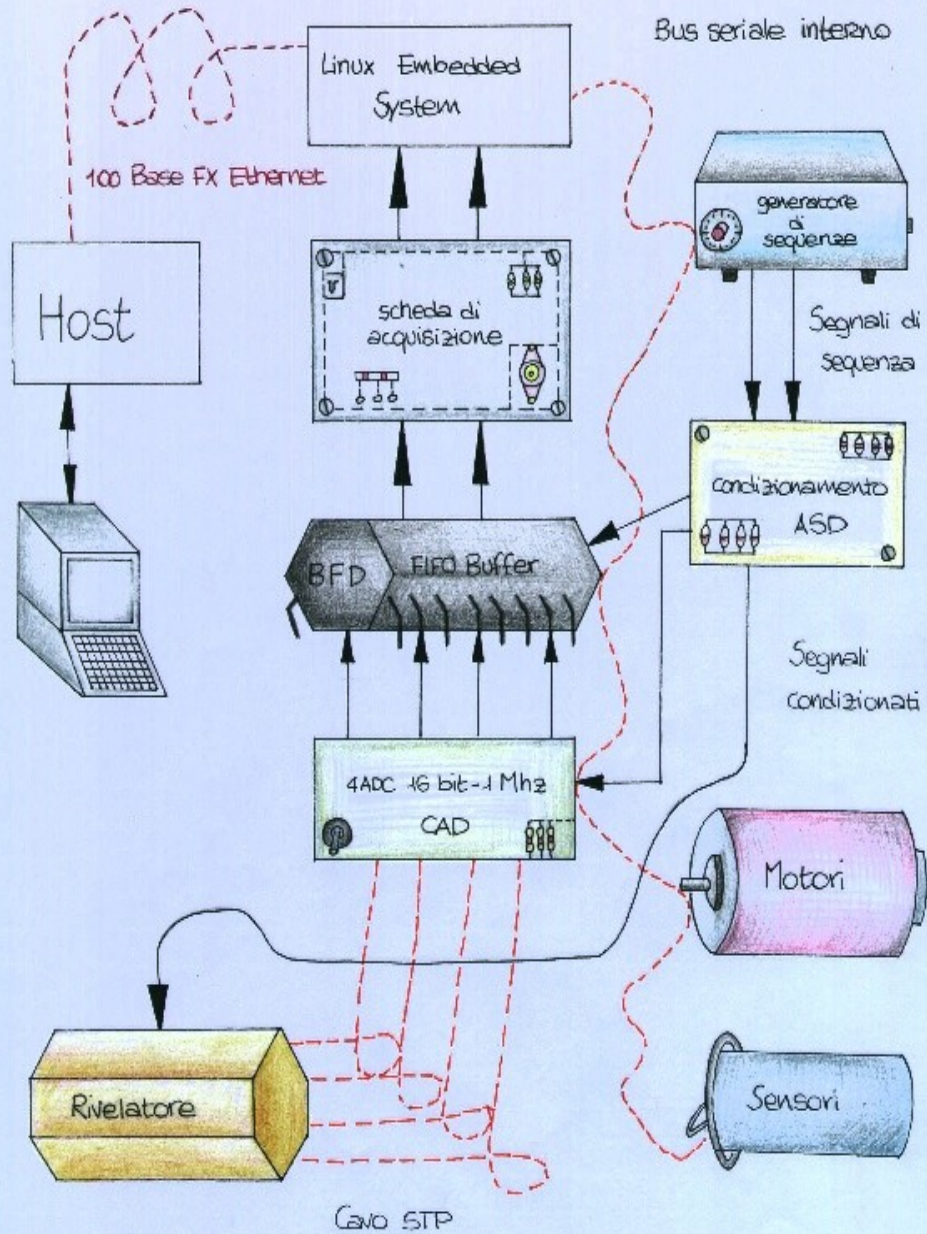
- reale natura real-time
- fortemente ottimizzati per il software e hardware embedded
- molti sistemi sviluppati in casa
- configurabili su misura
- codice aperto....
- a costo zero

Perché un sistema embedded

Fasti ovvero un nuovo modo di progettare sistema integrato per rivelatori astronomici

- ★ dividere il sistema in moduli con compiti specifici
- ★ utilizzare dispositivi di tipo commerciale
- ★ usare standard industriali come bus ISA, PCI, interfacce seriali etc
- ★ progettare dispositivi come concetti senza dipendere da un componente specifico

FASTI LAYOUT



*patent pending

ASPETTI HARDWARE

FASTI-NICS

- piattaforma Intel (ASUS)
- processore Celeron II (Coopermine) 800MHz
- 512 MB RAM
- assenza di drive mobili
- disco allo stato solido (Flash disk 256 MB)
- scheda di acquisizione PCI-DIO32HS 6533

FASTI-GIANO

- PC104 (PFM 620S)
- processore Intel Celeron 400 MHz
- 512 MB RAM
- disco allo stato solido memoria Compact Flash 1GB
- scheda di acquisizione custom ISA

Aspetti Software: perché Linux

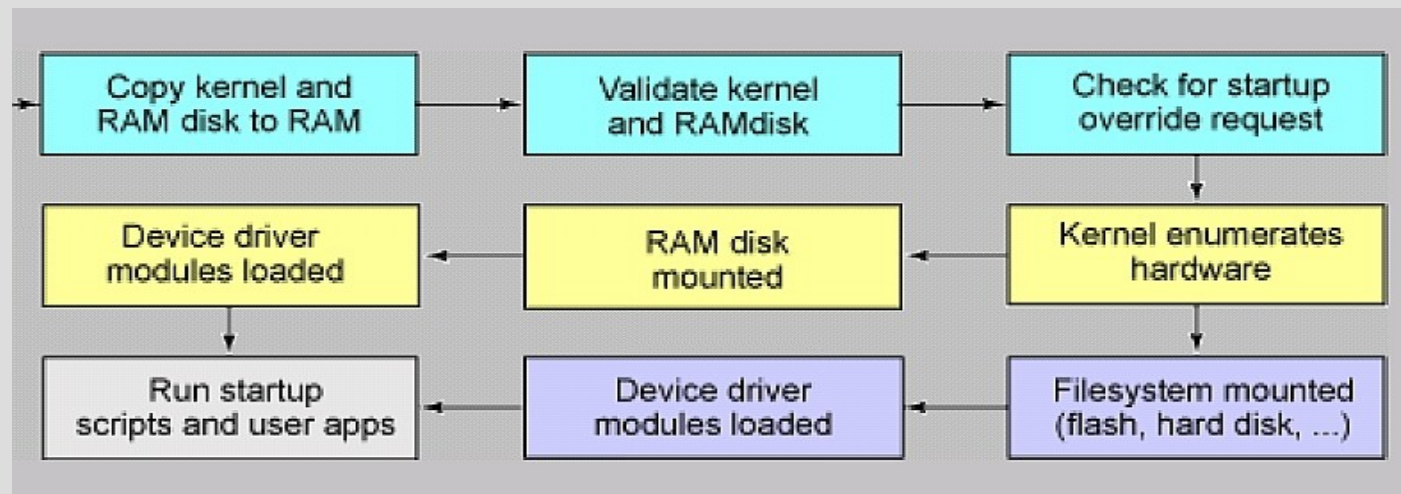
- perché no?
- non abbiamo vincoli operativi particolari
- possiamo trarre vantaggio da un SO
- buone prestazioni su hardware a basso costo
- flessibile, robusto
- supporto per molti dispositivi
- personalizzazione a basso livello del codice
- separazione spazio utente e spazio kernel
- vasta disponibilità di applicativi e librerie
- scalabilità su piattaforme diverse
- sistema standard per cui le conoscenze possono essere usate anche in ambito embedded

Di cosa abbiamo bisogno.....

- un host di sviluppo nell'architettura piu' conveniente
- un prototipo di host target (embedded)
- toolchain: insieme di eseguibili, librerie e compilatore per lo sviluppo del sistema
- bootloader per il target host
- un boot a doppio passo con il kernel caricato da un supporto esterno
- un kernel con la configurazione adeguata per il target
- libreria C per i due sistemi
- compilatore cross per l'host
- un root filesystem con i programmi e librerie fondamentali per il target (sistema base)
- un intero set delle librerie per le applicazioni per i due host

La fase di test viene effettuata sul sistema host

Il processo di boot



Se non diversamente configurato `initrd` viene decompresso e caricato in RAM il kernel lo monta come filesystem di root ed esegue lo script `linuxrc`. Quest'ultimo esegue tipicamente il `pivot_root` (cambio del filesystem di root) e passa il controllo a `init` da dove il run del sistema vero e proprio prosegue.

La maggior parte dei sistemi embedded prosegue semplicemente in RAM il proprio lavoro.

Sistema embedded per Nics

- sistema di sviluppo (HD da 850 MB) con tre partizioni + disco flash
- sistema di boot in due fasi (initrd)
- creazione del file ramdisk.img.gz e initrd.gz
- installazione Slackware minimale -> ramdisk.gz
- initrd di 4 MB con root fs minimale e applicativi delle binutils compilati staticamente
- Il disco flash al termine del boot non e' montato e il sistema lavora completamente in RAM

Il disco allo stato solido mantiene le informazioni persistenti:

- ★ aggiornamenti del kernel
- ★ aggiornamenti del root filesystem
- ★ informazioni di configurazione
- ★ dati di lavoro

Sistema embedded per Giano

- ♦ distribuzione SLAX (nata come LiveCD) installata sul disco CF
- ♦ è una collezione di moduli che sono archivi compressi caricati a run-time come fs read-only di tipo Squashfs
- ♦ Al boot i moduli selezionati sono aggiunti ad un filesystem di tipo Unionfs per fornire una vista unificata del filesystem.
- ♦ il ramo ad alta priorità di unionfs è un fs tmpfs
- ♦ la modularità dà origine a profili diversi di installazione

boot della SLAX

carica l'immagine initrd ed esegue linuxrc che:

- monta tmpfs su */memory* e crea */memory/changes*
- monta unionfs su */union* con il solo ramo **rw** */memory/changes*
- rilevamento dell'hardware e caricamento dei moduli dal disco
- i moduli montati in */memory/images* sono aggiunti all'unione in modalità *ro*
- viene eseguito *pivot_root* */union* diventa il nuovo root fs. La root originale viene mossa in */mnt/live*
- viene eseguito */sbin/init*
- con l'opzione di boot *copy2ram* i moduli al boot sono copiati in ram
- il disco CF viene smontato al termine del boot
- **il sistema è tutto in ram**
- la CF contiene il kernel, initrd i moduli la configurazione

UNIONFS

- ▶ UnionFS e' un filesystem Linux che simula l'unione di più filesystem sottostanti, mantenendone la semantica Unix
- ▶ sovrappone in modo trasparente file e cartelle di filesystem separati, detti *branch* (rami), per formare un singolo filesystem coerente.
- ▶ I vari rami possono essere filesystem ro o rw
- ▶ il filesystem UnionFS appare scrivibile: utile nei LiveCD
- ▶ Quando si aggiungono i rami, bisogna specificare la priorità
- ▶ viene visto solo il file nel ramo a priorità maggiore
- ▶ Le operazioni di scrittura in unionfs sono redirette verso uno specifico filesystem scrivibile