Sistemi Embedded

Sebbene i sistemi embedded siano in uso dagli anni '70, per la maggior parte della loro storia sono stati visti semplicemente come piccoli computer. Il principale problema ingegneristico era quello di far fronte a risorse limitate (potenza di elaborazione limitata, fonti di energia limitate, piccole memorie, ecc.). In quanto tale, la sfida ingegneristica era ottimizzare i progetti. Poiché tutti i progetti beneficiano dell'ottimizzazione, la disciplina non era distinta da nient'altro nell'informatica.
Di recente, la comunità ha capito che le principali sfide nei sistemi embedded derivano dalla loro interazione con i processi fisici e non dalle loro risorse limitate. Il termine sistemi cyber-fisici (CPS) è stato coniato da Helen Gill presso la National Science Foundation negli Stati Uniti per riferirsi all'integrazione della computazione con i processi fisici. In CPS, i computer e le reti embedded monitorano e controllano i processi fisici, di solito con circuiti di feedback in cui i processi fisici influiscono sui calcoli e viceversa. La progettazione di tali sistemi, quindi, richiede la comprensione delle dinamiche congiunte di computer, software, reti e processi fisici. È questo studio della dinamica articolare che contraddistingue questa disciplina.
Quando si studia il CPS, emergono alcuni problemi chiave che sono rari nel cosiddetto calcolo per scopi generici. Ad esempio, nel software generico, il tempo necessario per eseguire un'attività è una questione di prestazioni, non di correttezza. Non è sbagliato impiegare più tempo per eseguire un'attività. È semplicemente meno conveniente e quindi meno prezioso. In CPS, il tempo necessario per eseguire un'attività può essere fondamentale per il corretto funzionamento del sistema. Nel mondo fisico, a differenza del mondo cibernetico, il passare del tempo è inesorabile.
In CPS, inoltre, accadono molte cose contemporaneamente. I processi fisici sono composizioni di molte cose che accadono contemporaneamente, a differenza dei processi software, che sono profondamente radicati in passaggi sequenziali. Abelson e Sussman (1996) descrivono l'informatica come "epistemologia procedurale", conoscenza attraverso la procedura. Nel mondo fisico, al contrario, i processi sono raramente procedurali. I processi fisici sono composizioni di molti processi paralleli. Misurare e controllare la dinamica di questi processi orchestrando azioni che influenzano i processi sono i compiti principali dei sistemi embedded. Di conseguenza, la concorrenza è intrinseca in CPS. Molte delle sfide tecniche nella progettazione e nell'analisi del software embedded derivano dalla necessità di collegare una semantica intrinsecamente sequenziale con un mondo fisico intrinsecamente concorrente.
I progettisti di software embedded sono costretti a lottare con controller di interrupt, architetture di memoria, programmazione a livello di assembly (per sfruttare istruzioni specializzate o per controllare con precisione i tempi), progettazione di driver di dispositivo, interfacce di rete e strategie di pianificazione, piuttosto che concentrarsi sulla specifica del comportamento desiderato.
Sebbene queste tecnologie di implementazione siano (oggi) necessarie ai progettisti di sistemi per far funzionare i sistemi embedded, non costituiscono il nucleo intellettuale della disciplina. Il nucleo intellettuale è invece nei modelli e nelle astrazioni che congiungono computazione e dinamica fisica

1. Introduzione

Un sistema cyber-fisico (CPS) è un'integrazione del calcolo con processi fisici il cui comportamento è definito sia dalle parti cibernetiche che fisiche del sistema. I computer e le reti incorporati monitorano e controllano i processi fisici, di solito con circuiti di feedback in cui i processi fisici influiscono sui calcoli e viceversa. Come sfida intellettuale, CPS riguarda l'intersezione, non l'unione, del fisico e del cyber. Non è sufficiente comprendere separatamente le componenti fisiche e le componenti computazionali. Dobbiamo invece capire la loro interazione.

È facile immaginare l'utilizzo di CPS per migliorare molti sistemi esistenti, come i sistemi di produzione robotica; produzione e distribuzione di energia elettrica; controllo di processo negli stabilimenti chimici; giochi per computer distribuiti; trasporto di manufatti; riscaldamento, raffreddamento e illuminazione negli edifici; persone che si spostano come ascensori; e ponti che controllano il proprio stato di salute. L'impatto di tali miglioramenti sulla sicurezza, sul consumo di energia e sull'economia è potenzialmente enorme.
Molti degli esempi precedenti verranno implementati utilizzando una struttura come quella illustrata nella Figura 1.1. Ci sono tre parti principali in questo schizzo. In primo luogo, l'impianto fisico è la parte “fisica” di un sistema cyber-fisico. È semplicemente quella parte del sistema che non viene realizzata con i computer o le reti digitali. Può includere parti meccaniche, processi biologici o chimici o operatori umani. In secondo luogo, esistono una o più piattaforme computazionali, che consistono in sensori, attuatori, uno o più computer e (possibilmente) uno o più sistemi operativi. In terzo luogo, esiste un tessuto di rete, che fornisce i meccanismi per la comunicazione dei computer. Insieme, le piattaforme e il tessuto di rete costituiscono la parte “cyber” del sistema cyber-fisico.

1.3 Il processo di progettazione

L'obiettivo di questo libro è capire come progettare e implementare sistemi cyber-fisici. La Figura 1.3 mostra le tre parti principali del processo, modellazione, progettazione e analisi. La modellazione è il processo per acquisire una comprensione più profonda di un sistema attraverso l'imitazione. I modelli imitano il sistema e riflettono le proprietà del sistema. I modelli specificano cosa fa un sistema. Il design è la creazione strutturata di manufatti. Specifica come un sistema fa quello che fa. L'analisi è il processo per acquisire una comprensione più profonda di un sistema attraverso la dissezione. Specifica perché un sistema fa quello che fa (o non riesce a fare quello che un modello dice che dovrebbe fare).
In un processo di progettazione sano, l'analisi ha un ruolo di primo piano nelle prime fasi del processo. L'analisi sarà applicata ai modelli e ai disegni. I modelli possono essere analizzati per le condizioni di sicurezza, ad esempio per garantire un'invariante che affermi che se il veicolo si trova entro un metro dal suolo, la sua velocità verticale non è superiore a 0,1 metro/sec. I progetti possono essere analizzati per il comportamento temporale del software, ad esempio per determinare quanto tempo impiega il sistema per rispondere a un comando di arresto di emergenza.

1.3.1 Modellazione

Nella parte di modellazione del libro, definiamo un sistema come semplicemente una combinazione di parti che viene considerata nel suo insieme. Un sistema fisico è un sistema realizzato nella materia, in contrasto con un sistema concettuale o logico come software e algoritmi. La dinamica di un sistema è la sua evoluzione nel tempo: come cambia il suo stato. Un modello di un sistema fisico è una descrizione di alcuni aspetti del sistema che ha lo scopo di fornire informazioni sulle proprietà del sistema.
Un modello è esso stesso un sistema. È importante evitare di confondere un modello con il sistema che modella. Questi sono due manufatti distinti. Si dice che un modello di un sistema ha alta fedeltà se descrive accuratamente le proprietà del sistema. Si dice che astragga il sistema se omette i dettagli. I modelli dei sistemi fisici inevitabilmente omettono i dettagli, quindi sono sempre astrazioni del sistema.
Un sistema cyber-fisico (CPS) è un sistema composto da sottosistemi fisici insieme a elaborazione e rete. I modelli di sistemi cyber-fisici normalmente includono tutte e tre le parti. I modelli dovranno in genere rappresentare sia le proprietà dinamiche che statiche (quelle che non cambiano durante il funzionamento

del sistema). È importante notare che un modello di sistema cyber-fisico non deve necessariamente avere parti sia discrete che continue.

1.3.2 Progettazione

In tutti i capitoli della parte di progettazione, ci concentriamo in particolare sui meccanismi che forniscono concorrenza e controllo sui tempi, perché questi problemi incombono molto nella progettazione dei sistemi cyber-fisici.
Quando implementati in un prodotto, i processori embedded hanno in genere una funzione dedicata. Controllano un motore automobilistico o misurano lo spessore del ghiaccio nell'Artico. Non è loro richiesto di eseguire funzioni arbitrarie definite dall'utente.. Di conseguenza, i processori, le architetture di memoria, i meccanismi di I/O e i sistemi operativi possono essere più specializzati. Renderli più specializzati può portare enormi vantaggi.

1.3.3 Analisi

Ogni sistema deve essere progettato per soddisfare determinati requisiti. Per i sistemi embedded, che sono spesso destinati all'uso in applicazioni quotidiane critiche per la sicurezza, è essenziale certificare che il sistema soddisfi i suoi requisiti. Tali requisiti di sistema sono anche chiamati proprietà o specifiche.

7. Sensori ed attuatori

Un sensore è un dispositivo che misura una grandezza fisica. Un attuatore è un dispositivo che altera una grandezza fisica. Nei sistemi elettronici, i sensori producono spesso una tensione proporzionale alla grandezza fisica misurata. La tensione può quindi essere convertita in un numero da un convertitore analogico-digitale (ADC). Un sensore confezionato con un ADC è chiamato sensore digitale, mentre un sensore senza ADC è chiamato sensore analogico. Un sensore digitale avrà una precisione limitata, determinata dal numero di bit utilizzati per rappresentare il numero (può essere minimo uno!). Al contrario, un attuatore è comunemente azionato da una tensione che può essere convertita da un numero da un convertitore digitale-analogico (DAC). Un attuatore che è confezionato con un DAC è chiamato attuatore digitale.
Oggi, sensori e attuatori sono spesso confezionati con microprocessori e interfacce di rete, consentendo loro di apparire su Internet come servizi. La tendenza è verso una tecnologia che connetta profondamente il nostro mondo fisico con il nostro mondo dell'informazione attraverso sensori e attuatori così intelligenti. Questo mondo integrato è variamente chiamato Internet delle cose (IoT), Industria 4.0, Internet industriale, Machine-to-Machine (M2M), Internet of Everything, Smarter Planet, TSensors (Trillion Sensors) o The Fog (come The Cloud, ma più vicino al suolo).
Sono emerse alcune tecnologie per l'interfacciamento con sensori e attuatori che sfruttano meccanismi consolidati originariamente sviluppati per l'uso ordinario di Internet. Ad esempio, un sensore o un attuatore può essere accessibile tramite un server web utilizzando il cosiddetto stile architettonico Representational State Transfer (REST) (Fielding e Taylor, 2002). In questo stile, i dati possono essere recuperati da un sensore o comandi possono essere inviati a un attuatore costruendo un URL (Uniform Resource Locator), come se si accedesse a una normale pagina Web da un browser, e quindi trasmettendo l'URL direttamente al sensore o dispositivo attuatore o a un server web che funge da intermediario
In questo capitolo, non ci concentreremo su tali interfacce di alto livello, ma piuttosto sulle proprietà

fondamentali di sensori e attuatori come ponti tra il mondo fisico e quello cibernetico. Le principali proprietà di basso livello includono la velocità con cui vengono eseguite le misurazioni o le attuazioni, la costante di proporzionalità che mette in relazione la quantità fisica con la misurazione o il segnale di controllo, l'offset o la polarizzazione e la gamma dinamica. Per molti sensori e attuatori, è utile modellare il grado in cui un sensore o un attuatore si discosta da una misurazione proporzionale (la sua non linearità) e la quantità di variazione casuale introdotta dal processo di misurazione (il suo rumore).
Una preoccupazione fondamentale per sensori e attuatori è che il mondo fisico funzioni in un continuum multidimensionale di tempo e spazio. È un mondo analogico. Il mondo del software, invece, è digitale e rigorosamente quantizzato. Le misurazioni dei fenomeni fisici devono essere quantizzate sia in termini di grandezza che di tempo prima che il software possa operare su di essi. E anche i comandi al mondo fisico che provengono dal software saranno quantizzati intrinsecamente. Comprendere gli effetti di questa quantizzazione è essenziale.

7.1 Modelli di sensori ed attuatori

Sensori e attuatori collegano il mondo cibernetico con il mondo fisico.

7.1.1 Modelli lineari e affini

L'interpretazione delle letture di un tale sensore richiede la conoscenza della costante di proporzionalità e del bias. La costante di proporzionalità rappresenta la sensibilità del sensore, poiché specifica il grado di variazione della misura al variare della grandezza fisica.

7.1.2 Portata

Nessun sensore o attuatore realizza veramente una funzione affine. In particolare, la portata di un sensore, l'insieme dei valori di una grandezza fisica che può misurare, è sempre limitata.
Il sensore è ragionevolmente modellato da una funzione affine all'interno di un intervallo operativo (L, H), ma al di fuori di tale intervallo operativo, il suo comportamento è nettamente diverso.

7.1.3 Gamma dinamica

I sensori digitali non sono in grado di distinguere tra due valori ravvicinati della grandezza fisica. La precisione di un sensore è la più piccola differenza assoluta tra due valori di una grandezza fisica le cui letture del sensore sono distinguibili.

7.1.4 Quantizzazione

Un sensore digitale rappresenta una quantità fisica utilizzando un numero di n bit, dove n è un piccolo intero. Esistono solo 2 n numeri distinti di questo tipo, quindi un tale sensore può produrre solo 2 n misurazioni distinte. La quantità fisica effettiva può essere rappresentata da un numero reale x(t) ∈ R, ma per ciascuno di questi x(t), il sensore deve scegliere uno dei 2 n numeri per rappresentarlo. Questo processo è chiamato quantizzazione.

7.2.3 Misurazione della rotazione

Un giroscopio è un dispositivo che misura i cambiamenti di orientamento (rotazione). A differenza di un accelerometro, non è (per lo più) influenzato da un campo gravitazionale. I giroscopi tradizionali sono ingombranti dispositivi meccanici rotanti su un doppio supporto cardanico. I giroscopi moderni sono dispositivi MEMS (sistemi microelettromeccanici) che utilizzano piccole strutture risonanti, o dispositivi ottici che misurano la differenza di distanza percorsa da un raggio laser attorno a un percorso chiuso in direzioni opposte, o (per una precisione estremamente elevata) dispositivi che sfruttano effetti quantistici. Giroscopi e accelerometri possono essere combinati per migliorare l'accuratezza della navigazione inerziale, dove la posizione è stimata da dead reckoning. Chiamato anche ded reckoning (per calcolo dedotto), il dead reckoning inizia da una posizione e un orientamento iniziali noti e quindi utilizza le misurazioni del movimento per stimare la posizione e l'orientamento successivi. Un'unità di misura inerziale (IMU) o un sistema di navigazione inerziale (INS) utilizza un giroscopio per misurare i cambiamenti di orientamento e un accelerometro per misurare i cambiamenti di velocità. Tali unità sono soggette a deriva, ovviamente, quindi sono spesso combinate con unità GPS, che possono fornire periodicamente informazioni sulla posizione "nota buona" (sebbene non sull'orientamento). Le IMU possono diventare piuttosto sofisticate e costose.

7.2.4 Misurazione del suono

Un microfono misura le variazioni della pressione sonora.

7.2.5 Altri sensori

Un interruttore è un sensore particolarmente semplice. Progettato correttamente, può rilevare pressione, inclinazione o movimento, ad esempio, e spesso può essere collegato direttamente ai pin GPIO di un microcontrollore.

7.3 Attuatori

Come per i sensori, la varietà di attuatori disponibili è enorme. Dal momento che non possiamo fornire una copertura completa qui, discutiamo di due esempi comuni, LED e controllo del motore.

7.3.1 Diodi emettitori di luce

Pochissimi attuatori possono essere pilotati direttamente dai pin I/O digitali (pin GPIO) di un microcontrollore. Questi pin possono generare o assorbire una quantità limitata di corrente e qualsiasi tentativo di superare questa quantità rischia di danneggiare i circuiti. Un'eccezione sono i diodi a emissione di luce (LED), che, se messi in serie con un resistore, possono spesso essere collegati direttamente a un pin 200 GPIO. Ciò fornisce un modo conveniente per un sistema integrato per fornire un'indicazione visiva di alcune attività

7.3.2 Controllo motore

Un motore applica una coppia (forza angolare) a un carico proporzionale alla corrente attraverso gli avvolgimenti del motore. Potrebbe essere allettante, quindi, applicare una tensione al motore proporzionale alla coppia desiderata.
Utilizziamo una tecnica chiamata modulazione di larghezza di impulso (PWM), che può fornire in modo efficiente grandi quantità di potenza sotto controllo digitale, a condizione che il dispositivo a cui viene erogata la potenza possa tollerare l'accensione e lo spegnimento rapidi della fonte di alimentazione. I dispositivi che lo tollerano includono LED, lampade a incandescenza (è così che funzionano i dimmer) e motori CC. Un segnale PWM, come mostrato nella parte inferiore della Figura 7.5, passa da un livello alto a un livello basso a una frequenza specificata. Mantiene il segnale alto per una frazione del periodo di ciclo. Questa frazione è chiamata duty cycle e, nella Figura 7.5, è 0,1, o 10%. Un motore a corrente continua è costituito da un elettromagnete costituito da fili di avvolgimento attorno a un nucleo posto in un campo magnetico realizzato con magneti permanenti o elettromagneti. Quando la corrente scorre attraverso i fili, il nucleo gira. Un tale motore ha sia inerzia che induttanza che attenuano la sua risposta quando la corrente viene attivata e disattivata bruscamente, quindi tali motori tollerano bene i segnali PWM.

8. Embedded Processors

Nell'informatica generica, la varietà di architetture di set di istruzioni oggi è limitata, con l'architettura Intel x86 che domina in modo schiacciante tutto. Non esiste un tale predominio nell'informatica incorporata. Al contrario, la varietà di processori può essere scoraggiante per un progettista di sistema. Il nostro obiettivo in questo capitolo è fornire al lettore gli strumenti e il vocabolario per comprendere le opzioni e valutare criticamente le proprietà dei processori. Ci concentriamo in particolare sui meccanismi che forniscono concorrenza e controllo sui tempi, perché questi problemi incombono nella progettazione dei sistemi

cyber-fisici.
Quando implementati in un prodotto, i processori embedded hanno in genere una funzione dedicata. Controllano un motore automobilistico o misurano lo spessore del ghiaccio nell'Artico. Non viene loro chiesto di eseguire funzioni arbitrarie con il software definito dall'utente. Di conseguenza, i processori possono essere più specializzati. Renderli più specializzati può portare enormi vantaggi.
Quando si valutano i processori, è importante comprendere la differenza tra un'architettura del set di istruzioni (ISA) e la realizzazione di un processore o un chip. Quest'ultimo è un pezzo di silicio venduto da un venditore di semiconduttori. Il primo è una definizione delle istruzioni che il processore può eseguire e di alcuni vincoli strutturali (come la dimensione delle parole) che le realizzazioni devono condividere. x86 è un ISA. Ci sono molte realizzazioni. Un ISA è un'astrazione condivisa da molte realizzazioni. Un singolo ISA può apparire in molti chip diversi, spesso realizzati da produttori diversi e spesso con profili di prestazioni ampiamente variabili.
Il vantaggio di condividere un ISA in una famiglia di processori è che gli strumenti software, che sono costosi da sviluppare, possono essere condivisi e (a volte) gli stessi programmi possono essere eseguiti correttamente su più realizzazioni. Quest'ultima proprietà, tuttavia, è piuttosto infida, poiché un ISA normalmente non include alcun vincolo di tempistica. Quindi, sebbene un programma possa essere eseguito logicamente allo stesso modo su più chip, il comportamento del sistema potrebbe essere radicalmente diverso quando il processore è incorporato in un sistema cyber-fisico.

8.1 Tipi di processori

Come conseguenza dell'enorme varietà di applicazioni integrate, viene utilizzata un'enorme varietà di processori. Si va da dispositivi molto piccoli, lenti, poco costosi e a bassa potenza, a dispositivi ad alte prestazioni per scopi speciali. Questa sezione fornisce una panoramica di alcuni dei tipi di processori disponibili

8.1.1 Microcontrollori

Un microcontrollore (µC) è un piccolo computer su un unico circuito integrato costituito da un'unità di elaborazione centrale (CPU) relativamente semplice combinata con dispositivi periferici come memorie, dispositivi I/O e timer. Secondo alcuni, più della metà di tutte le CPU vendute nel mondo sono microcontrollori, sebbene un'affermazione del genere sia difficile da dimostrare perché la differenza tra microcontrollori e processori generici è indistinta. I microcontrollori più semplici funzionano su parole a 8 bit e sono adatti per applicazioni che richiedono piccole quantità di memoria e semplici funzioni logiche (rispetto alle funzioni aritmetiche ad alta intensità di prestazioni). Possono consumare quantità estremamente ridotte di energia e spesso includono una modalità di sospensione che riduce il consumo energetico a nanowatt. È stato dimostrato che componenti incorporati come nodi di rete di sensori e dispositivi di sorveglianza possono funzionare con una piccola batteria per diversi anni. I microcontrollori possono diventare piuttosto elaborati. Distinguerli dai processori generici può essere difficile. L'Intel Atom, ad esempio, è una famiglia di CPU x86 utilizzate principalmente nei netbook e in altri piccoli computer portatili. Poiché questi processori sono progettati per utilizzare relativamente poca energia senza perdere troppe prestazioni rispetto ai processori utilizzati nei computer di fascia alta, sono adatti per alcune applicazioni integrate e nei server in cui il raffreddamento è problematico. Geode di AMD è un altro esempio di processore vicino al confine sfocato tra processori generici e microcontrollori.

8.1.2 Processori DSP

Molte applicazioni embedded eseguono un bel po' di elaborazione del segnale. Un segnale è una raccolta di misurazioni campionate del mondo fisico, generalmente effettuate a una frequenza regolare chiamata frequenza di campionamento. Un'applicazione di controllo del movimento, ad esempio, può leggere le informazioni sulla posizione o sulla posizione dai sensori a frequenze di campionamento che vanno da pochi Hertz (Hz o campioni al secondo) a poche centinaia di Hertz.
I segnali audio vengono campionati a frequenze che vanno da 8.000 Hz (o 8 kHz, la frequenza di campionamento utilizzata nella telefonia per i segnali vocali) a 44,1 kHz (la frequenza di campionamento dei CD). Le applicazioni a ultrasuoni (come l'imaging medico) e le applicazioni musicali ad alte prestazioni possono campionare segnali sonori a velocità molto più elevate. Il video utilizza in genere frequenze di campionamento di 25 o 30 Hz per i dispositivi consumer a velocità molto più elevate per applicazioni di misurazione speciali.
Altre applicazioni integrate che fanno un uso massiccio dell'elaborazione del segnale includono giochi interattivi; sistemi di imaging radar, sonar e LIDAR (rilevamento e rilevamento della luce); video analytics (l'estrazione di informazioni da video, ad esempio per la sorveglianza); sistemi di assistenza alla guida per automobili; elettronica medica; e strumentazione scientifica.
Le applicazioni di elaborazione del segnale condividono tutte determinate caratteristiche. In primo luogo, gestiscono grandi quantità di dati. I dati possono rappresentare campioni nel tempo di un processore fisico (come campioni di un segnale radio wireless), campioni nello spazio (come immagini) o entrambi (come video e radar). In secondo luogo, in genere eseguono sofisticate operazioni matematiche sui dati, inclusi filtraggio, identificazione del sistema, analisi della frequenza, apprendimento automatico ed estrazione di funzionalità. Queste operazioni sono matematicamente intense. I processori progettati specificamente per supportare applicazioni di elaborazione del segnale ad alta intensità numerica sono chiamati processori DSP, o DSP (processori di segnale digitale), in breve. Per avere un'idea della struttura di tali processori e delle implicazioni per il progettista di software embedded, vale la pena comprendere la struttura dei tipici algoritmi di elaborazione del segnale. Un algoritmo canonico di elaborazione del segnale, utilizzato in qualche forma in tutte le applicazioni di cui sopra, è il filtraggio della risposta all'impulso finita (FIR). La forma più semplice di questo algoritmo è semplice, ma ha profonde implicazioni per l'hardware. In questa forma più semplice, un segnale di ingresso x è costituito da una lunghissima sequenza di valori numerici, tanto che ai fini della progettazione dovrebbe essere considerata infinita. Tale input può essere modellato come una funzione x: N → D, dove D è un insieme di valori in alcuni tipi di dati.1 Ad esempio, D potrebbe essere l'insieme di tutti gli interi a 16 bit, nel qual caso x(0) è il primo valore di input (un intero a 16 bit), x(1) è il secondo valore di input, ecc. Per comodità matematica, possiamo aumentare questo a x: Z → D definendo x(n) = 0 per tutti n < 0. Per ogni valore di ingresso x(n), un filtro FIR deve calcolare un valore di uscita y(n) secondo la formula,

$$y(n) = N\ X{-}1\ i{=}0\ aix(n - i)\ ,$$

DSP Processors

Architetture di computer specializzate per l'elaborazione del segnale esistono da un po' di tempo (Allen, 1975). I microprocessori DSP a chip singolo sono apparsi per la prima volta all'inizio degli anni '80, a cominciare dal Western Electric DSP1 di Bell Labs, l'S28211 di AMI, il TMS32010 di Texas Instruments, l'uPD7720 di NEC e pochi altri. Le prime applicazioni di questi dispositivi includevano modem dati voiceband, sintesi vocale, audio consumer, grafica e controller di unità disco. Una panoramica completa delle generazioni di processori DSP fino alla metà degli anni '90 può essere trovata in Lapsley et al. (1997). Le caratteristiche centrali dei DSP includono un'unità hardware ad accumulazione multipla; diverse varianti

dell'architettura di Harvard (per supportare più dati simultanei e recuperi di programmi); e modalità di indirizzamento che supportano l'incremento automatico, i buffer circolari e l'indirizzamento invertito (quest'ultimo per supportare il calcolo FFT). La maggior parte supporta la precisione dei dati in virgola fissa di 16-24 bit, in genere con accumulatori molto più ampi (40-56 bit) in modo che un gran numero di istruzioni di accumulazione multipla successive possano essere eseguite senza overflow. Sono apparsi alcuni DSP con hardware in virgola mobile, ma questi non hanno dominato il mercato.
I DSP sono difficili da programmare rispetto alle architetture RISC, principalmente a causa di istruzioni specializzate complesse, una pipeline esposta al programmatore e architetture di memoria asimmetriche. Fino alla fine degli anni '90, questi dispositivi erano quasi sempre programmati in linguaggio assembly. Ancora oggi, i programmi C fanno ampio uso di librerie che sono codificate a mano in linguaggio assembly per sfruttare le caratteristiche più esoteriche delle architetture.

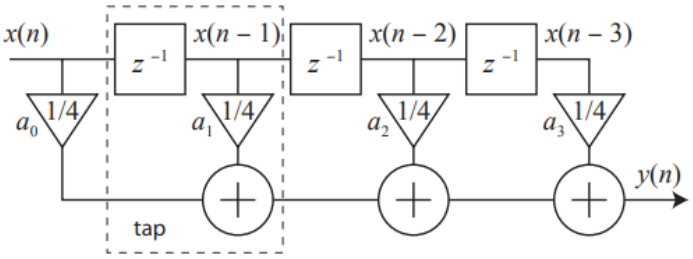


Figura 8.1: Struttura di un'implementazione di una linea di ritardo sfruttata del filtro FIR dell'esempio 8.1. Questo diagramma può essere letto come un diagramma del flusso di dati. Per ogni n ∈ N, ogni componente nel diagramma consuma un valore di input da ciascun percorso di input e produce un valore di output su ciascun percorso di output. Le caselle etichettate z −1 sono ritardi unitari. Il loro compito è produrre sul percorso di output il valore precedente dell'input (o un valore iniziale se non esisteva un input precedente). I triangoli moltiplicano il loro input per una costante e i cerchi aggiungono i loro input.
dove N è la lunghezza del filtro FIR e i coefficienti ai sono detti valori di tap. Puoi vedere da questa formula perché è utile aumentare il dominio della funzione x, poiché il calcolo di y(0), ad esempio, coinvolge valori x(−1), x(−2), ecc

Example 8.1: Supponiamo N = 4 e a0 = a1 = a2 = a3 = 1/4. Allora per ogni n ∈ N,

$$y(n) = (x(n) + x(n - 1) + x(n - 2) + x(n - 3))/4 .$$

Ciascun campione di output è la media dei quattro campioni di input più recenti. La struttura di questo calcolo è mostrata nella Figura 8.1. In quella figura, i valori di input entrano da sinistra e si propagano lungo la linea di ritardo, che viene toccata dopo ogni elemento di ritardo. Questa struttura è chiamata linea di ritardo tappata.

La frequenza con cui i valori di input x(n) vengono forniti e devono essere elaborati è chiamata frequenza di campionamento. Se si conosce la frequenza di campionamento e N, è possibile determinare il numero di operazioni aritmetiche che devono essere calcolate al secondo.

Esempio 8.2: Supponiamo che un filtro FIR sia dotato di campioni a una frequenza di 1 MHz (un milione di campioni al secondo) e che N = 32. Quindi le uscite devono essere calcolate a una frequenza di 1 MHz e ciascuna uscita richiede 32 moltiplicazioni e 31 aggiunte. Un processore deve essere in grado di sostenere una velocità di calcolo di 63 milioni di operazioni aritmetiche al secondo per implementare questa

applicazione. Naturalmente, per sostenere la velocità di calcolo, è necessario non solo che l'hardware aritmetico sia abbastanza veloce, ma anche che i meccanismi per ottenere dati dentro e fuori dalla memoria e dentro e fuori dal chip siano abbastanza veloci.

Un'immagine può essere modellata in modo simile come una funzione x: H × V → D, dove H ⊂ N rappresenta l'indice orizzontale, V ⊂ N rappresenta l'indice verticale e D è l'insieme di tutti i possibili valori di pixel. Un pixel (o elemento dell'immagine) è un campione che rappresenta il colore e l'intensità di un punto in un'immagine. Ci sono molti modi per farlo, ma tutti usano uno o più valori numerici per ogni pixel. Gli insiemi H e V dipendono dalla risoluzione dell'immagine.

Per un'immagine in scala di grigi, una tipica operazione di filtraggio costruirà una nuova immagine y da un'immagine originale x secondo la seguente formula,

$$\forall\, i \in H, j \in V, \quad y(i,j) = \sum_{n=-N}^{N} \sum_{m=-M}^{M} a_{n,m} x(i-n, j-m)\ ,$$

dove an,m sono i coefficienti di filtro. Questo è un filtro FIR bidimensionale. Tale calcolo richiede la definizione di x al di fuori della regione H × V . C'è una vera arte in questo (per evitare effetti di bordo), ma per i nostri scopi qui, è sufficiente avere un'idea della struttura del calcolo senza preoccuparsi di questo dettaglio. Un'immagine a colori avrà più canali di colore. Questi possono rappresentare la luminanza (quanto è luminoso il pixel) e la crominanza (qual è il colore del pixel), oppure possono rappresentare colori che possono essere composti per ottenere un colore arbitrario. In quest'ultimo caso, una scelta comune è un formato RGBA, che ha quattro canali che rappresentano il rosso, il verde, il blu e il canale alfa, che rappresenta la trasparenza. Ad esempio, un valore zero per R, G e B rappresenta il colore nero. Un valore zero per A rappresenta completamente trasparente (invisibile). Ogni canale ha anche un valore massimo, diciamo 1.0. Se tutti e quattro i canali sono al massimo, il colore risultante è un bianco completamente opaco. Il carico computazionale dell'operazione di filtraggio in (8.2) dipende dal numero di canali, dal numero di coefficienti di filtro (i valori di N e M), dalla risoluzione (le dimensioni degli insiemi H e V) e dal frame rate.
Oltre al gran numero di operazioni aritmetiche, il processore deve gestire il movimento dei dati lungo la linea di ritardo.

8.1.3 Processori grafici

Un'unità di elaborazione grafica (GPU) è un processore specializzato progettato appositamente per eseguire i calcoli richiesti nel rendering grafico. Tali processori risalgono agli anni '70, quando venivano utilizzati per il rendering di testo e grafica, per combinare più motivi grafici e per disegnare rettangoli, triangoli, cerchi e archi. Le moderne GPU supportano la grafica 3D, l'ombreggiatura e il video digitale. I fornitori dominanti di GPU oggi sono Intel, NVIDIA e AMD
Alcune applicazioni integrate, in particolare i giochi, si adattano bene alle GPU. Inoltre, le GPU si sono evolute verso modelli di programmazione più generali e quindi hanno iniziato ad apparire in altre applicazioni ad alta intensità di calcolo, come la strumentazione. Le GPU in genere sono piuttosto affamate di energia e quindi oggi non sono una buona scelta per le applicazioni embedded con limitazioni di energia.

8.2 Parallelism

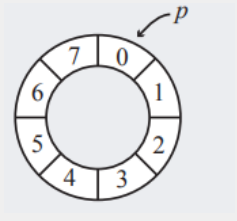
La maggior parte dei processori oggi fornisce varie forme di parallelismo. Questi meccanismi influenzano fortemente i tempi di esecuzione di un programma, quindi i progettisti di sistemi embedded devono comprenderli. Questa sezione fornisce una panoramica delle varie forme e delle loro conseguenze per i progettisti di sistemi.

8.2.1 Parallelism vs. Concurrency

La concorrenza è fondamentale per i sistemi embedded. Si dice che un programma per computer è simultaneo se diverse parti del programma vengono eseguite concettualmente contemporaneamente. Si dice che un programma è parallelo se parti diverse del programma vengono eseguite fisicamente contemporaneamente su hardware distinto (ad esempio su macchine multicore, su server in una server farm o su microprocessori distinti).

Buffer circolari

Un filtro FIR richiede una linea di ritardo come quella mostrata nella Figura 8.1. Un'implementazione ingenua allocherebbe un array in memoria e, ogni volta che arriva un campione di input, sposta ogni elemento dell'array nella posizione successiva più alta per fare spazio al nuovo elemento nella prima posizione. Questo sarebbe un enorme dispendio di larghezza di banda di memoria. Un approccio migliore consiste nell'utilizzare un buffer circolare, in cui un array in memoria viene interpretato come avente una struttura ad anello, come mostrato di seguito per una linea di ritardo di lunghezza 8:



Qui, 8 posizioni di memoria successive, etichettate da 0 a 7, memorizzano i valori nella linea di ritardo. Un puntatore p, inizializzato alla posizione 0, fornisce l'accesso. Un filtro FIR può utilizzare questo buffer circolare per implementare la somma di (8.1). Un'implementazione accetta prima un nuovo valore di input x(n), quindi calcola la somma all'indietro, iniziando con il termine i = N −1, dove nel nostro esempio, N = 8. Supponiamo che quando arriva l'n-esimo input, il valore di p è un numero pi ∈ {0, · · · , 7} (per il primo input x(0), pi = 0). Il programma scrive il nuovo input x(n) nella posizione data da p e quindi incrementa p, impostando p = pi + 1. Tutta l'aritmetica su p viene eseguita modulo 8, quindi ad esempio, se pi = 7, allora pi + 1 = 0. Il calcolo del filtro FIR legge quindi x(n − 7) dalla posizione p = pi + 1 e lo moltiplica per a7. Il risultato viene memorizzato in un registro accumulatore. Incrementa nuovamente p di uno, impostandolo a p = pi + 2. Quindi legge x(n − 6) dalla posizione p = pi + 2, lo moltiplica per a6 e aggiunge il risultato all'accumulatore (questo spiega il nome “accumulatore” per il registro, poiché accumula i prodotti nella linea di ritardo spillata). Continua finché non legge x(n) dalla posizione p = pi + 8, che a causa dell'operazione modulo è la stessa posizione in cui è stato scritto l'ultimo input x(n), e moltiplica quel valore per a0. Incrementa nuovamente p, ottenendo p = pi+9 = pi+1. Quindi, al termine di questa operazione, il valore di p è pi+1, che fornisce la posizione in cui deve essere scritto il prossimo input x(n+1).

I programmi non concorrenti specificano una sequenza di istruzioni da eseguire. Un linguaggio di programmazione che esprime un calcolo come una sequenza di operazioni è chiamato linguaggio

imperativo. C è un linguaggio imperativo. Quando si utilizza C per scrivere programmi simultanei, è necessario uscire dal linguaggio stesso, in genere utilizzando una libreria di thread. Una libreria di thread utilizza funzionalità fornite non da C, ma fornite dal sistema operativo e/o dall'hardware. Java è un linguaggio per lo più imperativo esteso con costrutti che supportano direttamente i thread. Pertanto, è possibile scrivere programmi simultanei in Java senza uscire dal linguaggio.

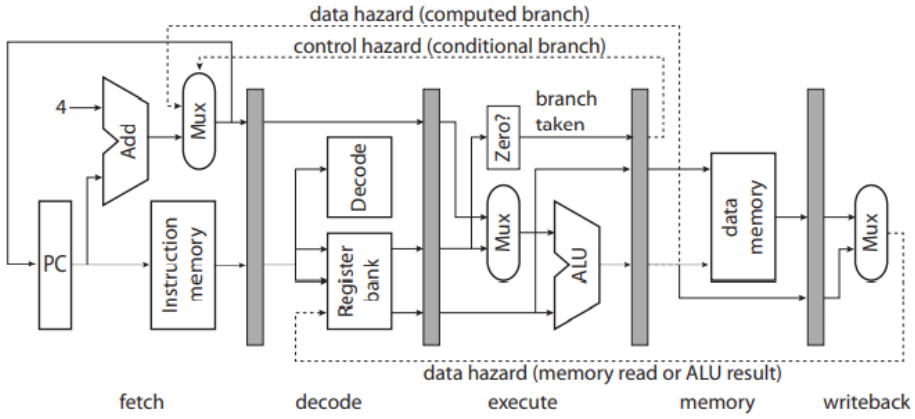
Ogni (corretta) esecuzione di un programma in un linguaggio imperativo deve comportarsi come se le istruzioni fossero eseguite esattamente nella sequenza specificata. Spesso, tuttavia, è possibile eseguire istruzioni in parallelo o in un ordine diverso da quello specificato dal programma e ottenere comunque un comportamento che corrisponda a quanto sarebbe accaduto se fossero state eseguite in sequenza.

Un compilatore può analizzare le dipendenze tra le operazioni in un programma e produrre codice parallelo, se la macchina di destinazione lo supporta. Questa analisi è chiamata analisi del flusso di dati. Molti microprocessori oggi supportano l'esecuzione parallela, utilizzando flussi di istruzioni multiproblemi o architetture VLIW (very large instruction word). I processori con flussi di istruzioni multiproblemi possono eseguire istruzioni indipendenti contemporaneamente. L'hardware analizza le istruzioni al volo per le dipendenze e, quando non c'è alcuna dipendenza, esegue più di un'istruzione alla volta. In quest'ultimo, le macchine VLIW hanno istruzioni a livello di assemblaggio che specificano più operazioni da eseguire insieme. In questo caso, il compilatore è solitamente tenuto a produrre le istruzioni parallele appropriate. In questi casi, l'analisi delle dipendenze viene eseguita a livello di linguaggio assembly oa livello di singole operazioni, non a livello di righe di C. Una riga di C può specificare operazioni multiple, o anche operazioni complesse come le chiamate di procedura. In entrambi i casi (multi-emissione e VLIW), un programma imperativo viene analizzato per la concorrenza al fine di consentire l'esecuzione parallela. L'obiettivo generale è accelerare l'esecuzione del programma. L'obiettivo è migliorare le prestazioni, in cui si presume che finire un compito prima sia sempre meglio che finirlo dopo.

Nel contesto dei sistemi embedded, tuttavia, la concorrenza gioca un ruolo molto più centrale del semplice miglioramento delle prestazioni. I programmi incorporati interagiscono con i processi fisici e, nel mondo fisico, molte attività progrediscono contemporaneamente. Un programma integrato spesso deve monitorare e reagire a più fonti di stimolo simultanee e controllare contemporaneamente più dispositivi di output che influenzano il mondo fisico. I programmi incorporati sono quasi sempre programmi concorrenti e la concorrenza è una parte intrinseca della logica dei programmi. Non è solo un modo per ottenere prestazioni migliori. In effetti, finire un compito prima non è necessariamente meglio che finirlo dopo. La tempestività conta, ovviamente; le azioni eseguite nel mondo fisico spesso devono essere eseguite al momento giusto (né presto né tardi). Immagina ad esempio un controller del motore per un motore a benzina. Accendere le candele prima non è certamente meglio che accenderle dopo. Devono essere licenziati al momento giusto.

Proprio come i programmi imperativi possono essere eseguiti in sequenza o in parallelo, i programmi concorrenti possono essere eseguiti in sequenza o in parallelo. L'esecuzione sequenziale di un programma simultaneo viene eseguita oggi in genere da un sistema operativo multitasking, che intercala l'esecuzione di più attività in un unico flusso sequenziale di istruzioni. Naturalmente, l'hardware può parallelizzare tale esecuzione se il processore ha un'architettura multi-problema o VLIW. Quindi, un programma simultaneo può essere convertito in un flusso sequenziale da un sistema operativo e di nuovo in un programma simultaneo dall'hardware, dove quest'ultima traduzione viene eseguita per migliorare le prestazioni. Queste traduzioni multiple complicano enormemente il problema di garantire che le cose accadano al momento giusto.

Il parallelismo nell'hardware, l'argomento principale di questo capitolo, esiste per migliorare le prestazioni per le applicazioni ad alta intensità di calcolo. Dal punto di vista del programmatore, la concorrenza nasce come conseguenza dell'hardware progettato per migliorare le prestazioni, non come conseguenza della risoluzione del problema dell'applicazione. In altre parole, l'applicazione non richiede (necessariamente) che più attività procedano contemporaneamente, richiede solo che le cose vengano eseguite molto rapidamente. Naturalmente, molte applicazioni interessanti uniranno entrambe le forme di concorrenza, derivanti dal parallelismo e dai requisiti dell'applicazione.
Il tipo di algoritmi che si trovano nei programmi embedded ad alta intensità di calcolo ha un profondo effetto sulla progettazione dell'hardware. In questa sezione, ci concentreremo sugli approcci hardware che forniscono il parallelismo, ovvero il pipelining, il parallelismo a livello di istruzione e le architetture multicore. Tutti hanno una forte influenza sui modelli di programmazione per il software embedded.



8.2.2 Pipelining

La maggior parte dei processori moderni sono pipeline. Una semplice pipeline a cinque stadi per una macchina a 32 bit è mostrata nella Figura 8.2. Nella figura, i rettangoli ombreggiati sono latch, che sono sincronizzati alla frequenza di clock del processore. Su ciascun fronte dell'orologio, il valore in ingresso viene memorizzato nel registro latch. L'uscita viene quindi mantenuta costante fino al bordo successivo dell'orologio, ai circuiti tra i latch di stabilizzarsi. Questo diagramma può essere visto come un modello sincronoreattivo del comportamento del processore.

Nella fase di recupero (più a sinistra) della pipeline, un contatore di programma (PC) fornisce un indirizzo alla memoria delle istruzioni. La memoria delle istruzioni nella figura fornisce codificate, che si presume siano larghe 32 bit. Nella fase di recupero, il PC viene incrementato di 4 (byte), per diventare l'indirizzo dell'istruzione successiva, a meno che un'istruzione di diramazione condizionale non fornisca un indirizzo completamente nuovo per il PC. La fase della pipeline di decodifica estrae gli indirizzi dei registri dall'istruzione a 32 bit e recupera i dati nei registri specificati dalla banca dei registri. La fase di esecuzione della pipeline opera sui dati prelevati dai registri o sul PC (per un ramo calcolato) utilizzando un'unità logica aritmetica (ALU), che esegue operazioni aritmetiche e logiche. La fase della pipeline di memoria legge o scrive in una posizione di memoria data da un registro. La fase della pipeline di writeback archivia i risultati nel file di registro.

Nella fase di recupero (più a sinistra) della pipeline, un contatore di programma (PC) fornisce un indirizzo alla memoria delle istruzioni. La memoria delle istruzioni fornisce istruzioni codificate, che nella figura si presume siano larghe 32 bit. Nella fase di recupero, il PC viene incrementato di 4 (byte), per diventare

l'indirizzo dell'istruzione successiva, a meno che un'istruzione di diramazione condizionale non fornisca un indirizzo completamente nuovo per il PC. La fase della pipeline di decodifica estrae gli indirizzi dei registri dall'istruzione a 32 bit e recupera i dati nei registri specificati dalla banca dei registri. La fase di esecuzione della pipeline opera sui dati prelevati dai registri o sul PC (per un ramo calcolato) utilizzando un'unità logica aritmetica (ALU), che esegue operazioni aritmetiche e logiche. La fase della pipeline di memoria legge o scrive in una posizione di memoria data da un registro. La fase della pipeline di writeback archivia i risultati nel file di registro.

Le porzioni di condotta tra i chiavistelli operano in parallelo. Quindi, possiamo vedere immediatamente che ci sono contemporaneamente cinque istruzioni in esecuzione, ciascuna in una diversa fase di esecuzione. Questo è facilmente visualizzabile con una tabella di prenotazione come quella della Figura 8.3. La tabella mostra le risorse hardware che possono essere utilizzate contemporaneamente a sinistra. In questo caso, la banca di registri appare tre volte perché la pipeline della Figura 8.2 presuppone che possano verificarsi due letture e scritture del file di registro in ogni ciclo.

La tabella di prenotazione nella Figura 8.3 mostra una sequenza A, B, C, D, E di istruzioni in un programma. Nel ciclo 5, E viene prelevato mentre D sta leggendo dal banco di registri, mentre C sta usando l'ALU, mentre B sta leggendo o scrivendo nella memoria dati, mentre A sta scrivendo i risultati nel banco di registri. La scrittura di A avviene nel ciclo 5, ma la lettura di B avviene nel ciclo 3. Pertanto, il valore che B legge non sarà il valore che A scrive.

Questo fenomeno è noto come rischio per i dati, una forma di rischio per le condotte. I rischi delle tubazioni sono causati dalle linee tratteggiate nella Figura 8.2. I programmatori normalmente si aspettano che se l'istruzione A è precedente all'istruzione B, tutti i risultati calcolati da A saranno disponibili per B, quindi questo comportamento potrebbe non essere accettabile.

Gli architetti informatici hanno affrontato il problema dei rischi delle condutture in vari modi. La tecnica più semplice è nota come pipeline esplicita. In questa tecnica, il rischio della pipeline è semplicemente documentato e il programmatore (o il compilatore) deve affrontarlo. Per l'esempio in cui B legge un registro scritto da A, il compilatore può inserire tre istruzioni no-op (che non fanno nulla) tra A e B per assicurarsi che la scrittura avvenga prima della lettura. Queste istruzioni no-op formano una bolla della pipeline che si propaga lungo la pipeline.

Una tecnica più elaborata consiste nel fornire interblocchi. In questa tecnica, l'hardware di decodifica delle istruzioni, incontrando l'istruzione B che legge un registro scritto da A, rileverà il pericolo e ritarderà l'esecuzione di B finché A non avrà completato la fase di writeback. Per questa pipeline, B dovrebbe essere ritardato di tre cicli di clock per consentire ad A di completare, come mostrato nella Figura 8.4. Questo può essere ridotto a due cicli se viene fornita una logica di inoltro leggermente più complessa, che rileva che A sta scrivendo la stessa posizione che B sta leggendo e fornisce direttamente i dati anziché richiedere che la scrittura avvenga prima della lettura. Gli interblocchi forniscono quindi hardware che inserisce automaticamente le bolle della pipeline.

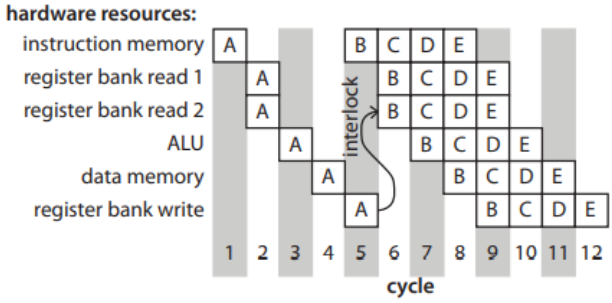
Una tecnica ancora più elaborata è l'esecuzione fuori ordine, in cui viene fornito hardware che rileva un pericolo, ma invece di ritardare semplicemente l'esecuzione di B, procede a recuperare C e se C non legge i registri scritti da A o B, e non scrive i registri letti da B, quindi procede all'esecuzione di C prima di B. Ciò riduce ulteriormente il numero di bolle della pipeline.

Un'altra forma di pericolo di gasdotto illustrata nella Figura 8.2 è un pericolo di controllo. Nella figura, un'istruzione di diramazione condizionale cambia il valore del PC se un registro specificato ha valore zero. Il

nuovo valore del PC è fornito (opzionalmente) dal risultato di un'operazione ALU. In questo caso, se A è un'istruzione di diramazione condizionale, allora A deve aver raggiunto lo stadio di memoria prima che il PC possa essere aggiornato. Le istruzioni che seguono A in memoria saranno state recuperate e saranno già nelle fasi di decodifica ed esecuzione nel momento in cui verrà determinato che tali istruzioni non dovrebbero essere effettivamente eseguite.

Come i rischi relativi ai dati, esistono diverse tecniche per affrontare i rischi di controllo. Un ramo ritardato documenta semplicemente il fatto che il ramo verrà preso un certo numero di cicli dopo che è stato incontrato e lascia al programmatore (o compilatore) il compito di assicurarsi che le istruzioni che seguono l'istruzione del ramo condizionale siano innocue (come no -ops) o fare un lavoro utile che non dipende dal fatto che il ramo sia stato preso. Un interblocco fornisce l'hardware per inserire bolle di pipeline secondo necessità, proprio come con i rischi relativi ai dati. Nella tecnica più elaborata, l'esecuzione speculativa, l'hardware stima se è probabile che il ramo venga preso e inizia a eseguire le istruzioni che si aspetta di eseguire. Se la sua aspettativa non viene soddisfatta, annulla tutti gli effetti collaterali (come le scritture dei registri) causati dalle istruzioni eseguite in modo speculativo.

Fatta eccezione per pipeline esplicite e diramazioni ritardate, tutte queste tecniche introducono variabilità nei tempi di esecuzione di una sequenza di istruzioni. L'analisi della tempistica di un programma può diventare estremamente difficile quando esiste una pipeline profonda con inoltro e speculazioni elaborati. Le pipeline esplicite sono relativamente comuni nei processori DSP, che vengono spesso applicati in contesti in cui è essenziale una tempistica precisa. L'esecuzione fuori ordine e speculativa è comune nei processori generici, dove la tempistica conta solo in senso aggregato. Un progettista di sistemi embedded deve comprendere i requisiti dell'applicazione ed evitare processori in cui il livello richiesto di precisione di temporizzazione è irraggiungibile.



8.2.3 Instruction-Level Parallelism

Il raggiungimento di prestazioni elevate richiede parallelismo nell'hardware. Tale parallelismo può assumere due forme generali, le architetture multicore, descritte più avanti nella Sezione 8.2.4, o il parallelismo a livello di istruzione (ILP), che è l'argomento di questa sezione. Un processore che supporta ILP è in grado di eseguire più operazioni indipendenti in ogni ciclo di istruzione. Discutiamo di quattro forme principali di ILP: istruzioni CISC, parallelismo delle sottoparole, superscalare e VLIW.

CISC Instructions

Un processore con istruzioni complesse (e in genere piuttosto specializzate) è chiamato macchina CISC (computer con set di istruzioni complesse). La filosofia alla base di tali processori è nettamente diversa da

quella delle macchine RISC (computer con set di istruzioni ridotte) (Patterson e Ditzel, 1980). I DSP sono tipicamente macchine CISC e includono istruzioni che supportano specificamente il filtraggio FIR (e spesso altri algoritmi come FFT (trasformate veloci di Fourier) e decodifica di Viterbi). Infatti, per qualificarsi come DSP, un processore deve essere in grado di eseguire il filtraggio FIR in un ciclo di istruzioni per tap

Example 8.6: La famiglia di processori DSP TMS320c54x di Texas Instruments è concepita per essere utilizzata in applicazioni embedded con limitazioni di potenza che richiedono elevate prestazioni di elaborazione del segnale, come sistemi di comunicazione wireless e PDA (Personal Digital Assistant). Il ciclo interno di un calcolo FIR (8.1) è

```
1 RPT numberOfTaps - 1
2 MAC *AR2+, *AR3+, A
```

La prima istruzione illustra i cicli zero-overhead che si trovano comunemente nei DSP. L'istruzione successiva verrà eseguita un numero di volte pari a uno più l'argomento dell'istruzione RPT. L'istruzione MAC è un'istruzione multiplyaccumulate, prevalente anche nelle architetture DSP. Ha tre argomenti che specificano il seguente calcolo,

$$a := a + x * y ,$$

dove a è il contenuto di un registro accumulatore denominato A e xey sono valori trovati in memoria. Gli indirizzi di questi valori sono contenuti nei registri ausiliari AR2 e AR3. Questi registri vengono incrementati automaticamente dopo l'accesso. Inoltre, questi registri possono essere impostati per implementare buffer circolari, come descritto nel riquadro a pagina 221. Il processore c54x include una sezione di memoria on-chip che supporta due accessi in un unico ciclo e purché gli indirizzi si riferiscano a in questa sezione della memoria, l'istruzione MAC verrà eseguita in un unico ciclo. Pertanto, ad ogni ciclo, il processore esegue due recuperi di memoria, una moltiplicazione, un'addizione ordinaria e due incrementi di indirizzo (possibilmente modulo). Tutti i DSP hanno capacità simili.

Le istruzioni CISC possono diventare piuttosto esoteriche.

Esempio 8.7: I coefficienti del filtro FIR in (8.1) sono spesso simmetrici, il che significa che N è pari e

$$ai = aN−i−1 .$$

La ragione di ciò è che tali filtri hanno una fase lineare (intuitivamente, ciò significa che i segnali di ingresso simmetrici producono segnali di uscita simmetrici o che tutte le componenti di frequenza sono ritardate della stessa quantità). In questo caso, possiamo ridurre il numero di moltiplicazioni riscrivendo (8.1) come

$$y(n) = \sum_{i=0}^{(N/2)-1} a_i(x(n-i) + x(n-N+i+1)) .$$

Il set di istruzioni TMS320c54x di Texas Instruments include un'istruzione FIRS che funziona in modo simile al MAC nell'Esempio 8.6, ma utilizza questo calcolo anziché quello di (8.1). Questo sfrutta il fatto che il c54x ha due ALU, e quindi può fare il doppio delle addizioni rispetto alle moltiplicazioni. Il tempo per eseguire un filtro FIR ora si riduce a 1/2 ciclo per tap.

I set di istruzioni ISC hanno i loro svantaggi. Per uno, è estremamente difficile (forse impossibile) per un compilatore fare un uso ottimale di un tale set di istruzioni. Di conseguenza, i processori DSP sono comunemente usati con librerie di codice scritte e ottimizzate in linguaggio assembly. Inoltre, i set di istruzioni CISC possono presentare sottili problemi di temporizzazione che possono interferire con il raggiungimento di una pianificazione difficile in tempo reale.

Subword Parallelism

Molte applicazioni integrate operano su tipi di dati notevolmente inferiori alla dimensione della parola del processore.
Per supportare tali tipi di dati, alcuni processori supportano il parallelismo delle sottoparole, in cui un'ampia ALU è divisa in sezioni più strette consentendo operazioni aritmetiche o logiche simultanee su parole più piccole.

Esempio 8.9: Intel ha introdotto il parallelismo delle sottoparole nel processore Pentium per uso generico ampiamente utilizzato e ha chiamato la tecnologia MMX (Eden e Kagan, 1997). Le istruzioni MMX dividono il percorso dati a 64 bit in sezioni di appena 8 bit, supportando operazioni identiche simultanee su più byte di dati pixel dell'immagine. La tecnologia è stata utilizzata per migliorare le prestazioni delle applicazioni di manipolazione delle immagini e delle applicazioni che supportano lo streaming video. Tecniche simili sono state introdotte da Sun Microsystems per i processori SparcTM (Tremblay et al., 1996) e da Hewlett Packard per il processore PA RISC (Lee, 1996). Molte architetture di processori progettate per applicazioni integrate, inclusi molti processori DSP, supportano anche il parallelismo delle sottoparole.

Un processore vettoriale è quello in cui il set di istruzioni include operazioni su più elementi di dati contemporaneamente. Il parallelismo delle sottoparole è una forma particolare di elaborazione vettoriale.

Superscalar

I processori superscalari utilizzano set di istruzioni sequenziali abbastanza convenzionali, ma l'hardware può inviare simultaneamente più istruzioni a unità hardware distinte quando rileva che tale invio simultaneo non cambierà il comportamento del programma. Cioè, l'esecuzione del programma è identica a quella che sarebbe stata se fosse stato eseguito in sequenza. Tali processori supportano anche l'esecuzione fuori ordine, in cui le istruzioni successive nel flusso vengono eseguite prima delle istruzioni precedenti. I processori superscalari presentano uno svantaggio significativo per i sistemi embedded, ovvero che i tempi di esecuzione possono essere estremamente difficili da prevedere e, nel contesto del multitasking (interrupt e thread), potrebbero non essere nemmeno ripetibili. I tempi di esecuzione possono essere molto sensibili all'esatta tempistica degli interrupt, in quanto piccole variazioni di tale tempistica possono avere grandi effetti sui tempi di esecuzione dei programmi.

VLIW

I processori destinati alle applicazioni embedded utilizzano spesso architetture VLIW anziché superscalari per ottenere tempi più ripetibili e prevedibili. I processori VLIW (very large in struction word) includono più unità di funzione, come i processori superscalari, ma invece di determinare dinamicamente quali istruzioni possono essere eseguite contemporaneamente, ciascuna istruzione specifica cosa dovrebbe fare ogni unità di funzione in un ciclo particolare. Cioè, un set di istruzioni VLIW combina più operazioni indipendenti in un'unica istruzione. Come le architetture superscalari, queste operazioni multiple vengono eseguite

contemporaneamente su hardware distinto. A differenza del superscalare, tuttavia, l'ordine e la simultaneità dell'esecuzione sono fissi nel programma anziché essere decisi al volo. Spetta al programmatore (che lavora a livello di linguaggio assembly) o al compilatore garantire che le operazioni simultanee siano effettivamente indipendenti. In cambio di questa ulteriore complessità nella programmazione, i tempi di esecuzione diventano ripetibili e (spesso) prevedibili.

Esempio 8.10: Nell'esempio 8.7, abbiamo visto l'istruzione specializzata FIRS dell'architettura c54x che specifica le operazioni per due ALU e un moltiplicatore. Questa può essere considerata una forma primitiva di VLIW, ma le generazioni successive di processori sono molto più esplicite sulla loro natura VLIW. Il Texas Instruments TMS320c55x, la generazione successiva oltre al c54x, include due unità ad accumulo multiplo e può supportare istruzioni simili a questa:

```
 MAC *AR2+, *CDP+, AC0
:: MAC *AR3+, *CDP+, AC1
```

Qui, AC0 e AC1 sono due registri accumulatori e CDP è un registro specializzato per indicare i coefficienti di filtro. La notazione :: significa che queste due istruzioni devono essere emesse ed eseguite nello stesso ciclo. Spetta al programmatore o al compilatore determinare se queste istruzioni possono effettivamente essere eseguite contemporaneamente. Supponendo che gli indirizzi di memoria siano tali che i recuperi possano avvenire contemporaneamente, queste due istruzioni MAC vengono eseguite in un unico ciclo, dividendo effettivamente in metà il tempo necessario per eseguire un filtro FIR.

Per le applicazioni che richiedono prestazioni ancora più elevate, le architetture VLIW possono diventare piuttosto elaborate.

8.2.4 Multicore Architectures

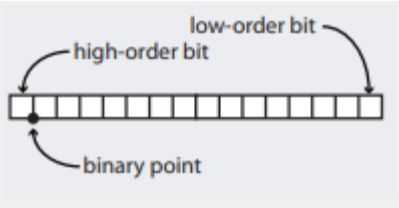
Una macchina multicore è una combinazione di più processori su un singolo chip. Sebbene le macchine multicore esistano dall'inizio degli anni '90, sono entrate solo di recente nell'informatica generica. Questa penetrazione rappresenta oggi gran parte dell'interesse per loro. Le macchine multicore eterogenee combinano una varietà di tipi di processore su un singolo chip, rispetto a più istanze dello stesso tipo di processore.

Esempio 8.13: Le architetture OMAP (piattaforma di applicazioni multimediali aperte) di Texas Instruments sono ampiamente utilizzate nei telefoni cellulari, che normalmente combinano uno o più processori DSP con uno o più processori più simili nello stile ai processori generici. I processori DSP gestiscono l'elaborazione radio, vocale e multimediale (audio, immagini e video). Gli altri processori gestiscono l'interfaccia utente, le funzioni del database, la rete e le applicazioni scaricabili. In particolare, l'OMAP4440 include un processore ARM Cortex dual-core da 1 GHz, un DSP c64x, una GPU e un processore di segnale immagine.

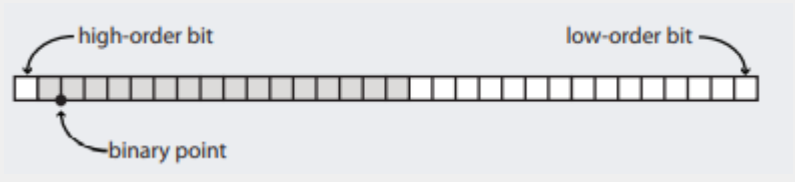
Fixed-Point Numbers

Molti processori embedded forniscono hardware solo per l'aritmetica intera. L'aritmetica degli interi, tuttavia, può essere utilizzata per i numeri non interi, con una certa attenzione. Dato, ad esempio, un numero intero di 16 bit, un programmatore può immaginare un punto binario, che è come un punto

decimale, tranne per il fatto che separa i bit anziché le cifre del numero. Ad esempio, un intero a 16 bit può essere utilizzato per rappresentare numeri nell'intervallo da -1,0 a 1,0 (approssimativamente) posizionando un punto binario (concettuale) appena sotto il bit di ordine superiore del numero, come mostrato di seguito:



Senza il punto binario, un numero rappresentato dai 16 bit è un numero intero $x \in \{-2^{15}, \cdots, 2^{15}-1\}$ (assumendo la rappresentazione binaria del complemento a due, che è diventata quasi universale per gli interi con segno). Con il punto binario, interpretiamo i 16 bit per rappresentare un numero $y = x/2^{15}$. Quindi, y varia da −1 a $1-2^{-15}$. Questo è noto come numero a virgola fissa. Il formato di questo numero a virgola fissa può essere scritto 1.15, indicando che c'è un bit a sinistra del punto binario e 15 a destra. Quando due di questi numeri vengono moltiplicati alla massima precisione, il risultato è un numero a 32 bit. Il punto binario si trova come segue:



La posizione del punto binario segue dalla legge di conservazione dei bit. Quando si moltiplicano due numeri con i formati n.m e p.q, il risultato ha il formato (n+p).(m+ q). I processori spesso supportano tali moltiplicazioni ad alta precisione, in cui il risultato va in un registro accumulatore che ha almeno il doppio dei bit dei normali registri di dati. Per riscrivere il risultato in un registro dati, tuttavia, dobbiamo estrarre 16 bit dal risultato a 32 bit. Se estraiamo i bit ombreggiati a pagina 235, conserviamo la posizione del punto binario e il risultato rappresenta ancora un numero all'incirca compreso tra -1 e 1. C'è una perdita di informazioni, tuttavia, quando estraiamo 16 bit da un risultato a 32 bit. In primo luogo, esiste la possibilità di overflow, perché stiamo scartando il bit di ordine superiore. Supponiamo che i due numeri da moltiplicare siano entrambi −1, che ha una rappresentazione binaria in complemento a due come segue:

Quando questi due numeri vengono moltiplicati, il risultato ha il seguente schema di bit:

che in complemento a due rappresenta 1, il risultato corretto. Tuttavia, quando estraiamo i 16 bit ombreggiati, il risultato è ora -1! Infatti, 1 non è rappresentabile nel formato a virgola fissa 1.15, quindi si è verificato un overflow. I programmatori devono proteggersi da questo, ad esempio assicurandosi che tutti i numeri siano rigorosamente inferiori a 1 in grandezza, vietando -1. Un secondo problema è che quando estraiamo i 16 bit ombreggiati da un risultato a 32 bit, scartiamo 15 bit di ordine inferiore. C'è una perdita di informazioni qui. Se scartiamo semplicemente i 15 bit di ordine inferiore, la strategia è nota come troncamento. Se invece aggiungiamo prima il seguente schema di bit al risultato a 32 bit, il risultato è noto come arrotondamento:

L'arrotondamento sceglie il risultato più vicino al risultato di precisione completa, mentre il troncamento sceglie il risultato più vicino di magnitudine inferiore. I processori DSP in genere eseguono l'estrazione di cui

sopra con arrotondamento o troncamento nell'hardware quando i dati vengono spostati da un accumulatore a un registro generico o alla memoria.

Per le applicazioni embedded, le architetture multicore presentano un potenziale vantaggio significativo rispetto alle architetture single-core perché le attività in tempo reale e critiche per la sicurezza possono avere un processore dedicato. Questo è il motivo delle architetture eterogenee utilizzate per i telefoni cellulari, poiché le funzioni di elaborazione radio e vocale sono funzioni hard real-time con un notevole carico computazionale. In tali architetture, le applicazioni utente non possono interferire con le funzioni in tempo reale. Questa mancanza di interferenza è più problematica nelle architetture multicore per uso generale. È comune, ad esempio, utilizzare cache multilivello, in cui la cache di secondo livello o superiore è condivisa tra i core. Sfortunatamente, tale condivisione rende molto difficile isolare il comportamento in tempo reale dei programmi su core separati, poiché ogni programma può innescare cache miss in un altro core. Tali cache multilivello non sono adatte per applicazioni in tempo reale. Un tipo molto diverso di architettura multicore che viene talvolta utilizzata nelle applicazioni embedded utilizza uno o più soft core insieme a hardware personalizzato su un FPGA (field-programmable gate array). Gli FPGA sono chip la cui funzione hardware è programmabile utilizzando strumenti di progettazione hardware. I soft core sono processori implementati su FPGA. Il vantaggio dei soft core è che possono essere accoppiati strettamente all'hardware personalizzato più facilmente rispetto ai processori standard.

9. Memory Architecture

Molti architetti di processori sostengono che i sistemi di memoria hanno un impatto maggiore sulle prestazioni complessive del sistema rispetto alle pipeline di dati. Questo dipende, ovviamente, dall'applicazione, ma per molte applicazioni è vero. Ci sono tre principali fonti di complessità nella memoria. In primo luogo, è comunemente necessario combinare una varietà di tecnologie di memoria nello stesso sistema embedded. Molte tecnologie di memoria sono volatili, il che significa che il contenuto della memoria viene perso in caso di interruzione dell'alimentazione. La maggior parte dei sistemi embedded richiede almeno un po' di memoria non volatile e un po' di memoria volatile. Inoltre, all'interno di queste categorie, ci sono diverse scelte e le scelte hanno conseguenze significative per il progettista del sistema. In secondo luogo, la gerarchia della memoria è spesso necessaria perché le memorie con maggiore capacità e/o minore consumo energetico sono più lente. Per ottenere prestazioni ragionevoli a costi ragionevoli, le memorie più veloci devono essere mescolate con le memorie più lente. In terzo luogo, lo spazio degli indirizzi di un'architettura di processore è suddiviso per fornire accesso ai vari tipi di memoria, per fornire supporto per modelli di programmazione comuni e per designare indirizzi per l'interazione con dispositivi diversi dalle memorie, come i dispositivi I/O.

9.1 Memory Technologies

Nei sistemi embedded, i problemi di memoria incombono. Le scelte delle tecnologie di memoria hanno importanti conseguenze per il progettista del sistema. Ad esempio, un programmatore potrebbe dover preoccuparsi se i dati persisteranno quando l'alimentazione viene spenta o viene attivata una modalità standby di risparmio energetico. Una memoria il cui contenuto va perso quando viene interrotta l'alimentazione è chiamata memoria volatile.

9.1.1 RAM

Oltre al file di registro, un microcomputer include tipicamente una certa quantità di RAM (memoria ad accesso casuale), che è una memoria in cui i singoli elementi (byte o parole) possono essere scritti e letti uno alla volta in modo relativamente rapido. La SRAM (RAM statica) è più veloce della DRAM (RAM dinamica), ma è anche più grande (ogni bit occupa più area di silicio). La DRAM conserva i dati solo per un breve periodo, quindi ogni posizione di memoria deve essere periodicamente aggiornata. La SRAM conserva i dati per tutto il tempo in cui viene mantenuta l'alimentazione. Entrambi i tipi di memorie perdono il loro contenuto in caso di interruzione dell'alimentazione, quindi entrambi sono memoria volatile, anche se probabilmente la DRAM è più volatile della SRAM perché perde il suo contenuto anche se viene mantenuta l'alimentazione.
La maggior parte dei sistemi informatici incorporati include una memoria SRAM. Molti includono anche la DRAM perché può essere poco pratico fornire memoria sufficiente con la sola tecnologia SRAM. Un programmatore preoccupato del tempo necessario per l'esecuzione di un programma deve sapere se gli indirizzi di memoria a cui si accede sono mappati su SRAM o DRAM. Inoltre, il ciclo di aggiornamento della DRAM può introdurre variabilità nei tempi di accesso perché la DRAM potrebbe essere occupata con un aggiornamento nel momento in cui viene richiesto l'accesso. Inoltre, la cronologia degli accessi può influire sui tempi di accesso. Il tempo necessario per accedere a un indirizzo di memoria può dipendere dall'ultimo indirizzo di memoria a cui è stato effettuato l'accesso.
Un produttore di un chip di memoria DRAM specificherà che ogni posizione di memoria deve essere aggiornata, ad esempio ogni 64 ms, e che un certo numero di posizioni (una "riga") vengono aggiornate insieme.

9.1.2 Non-Volatile Memory

I sistemi embedded devono invariabilmente memorizzare i dati anche quando l'alimentazione è spenta. Ci sono diverse opzioni per questo. Uno, ovviamente, è fornire una batteria di riserva in modo che l'energia non vada mai persa. Le batterie, tuttavia, si consumano e sono disponibili opzioni migliori, note collettivamente come memorie non volatili. Una prima forma di memoria non volatile era la memoria del nucleo magnetico o semplicemente il nucleo, in cui un anello ferromagnetico veniva magnetizzato per memorizzare i dati. Il termine "core" persiste nell'informatica per riferirsi alle memorie dei computer, sebbene questo possa cambiare man mano che le macchine multicore diventano onnipresenti.
La memoria non volatile più basilare oggi è la ROM (memoria di sola lettura) o la ROM maschera, il cui contenuto è riparato in fabbrica. Questo può essere utile per i prodotti prodotti in serie che necessitano solo di un programma e di dati costanti memorizzati e questi dati non cambiano mai. Tali programmi sono noti come firmware, suggerendo che non sono "soft" come il software. Esistono diverse varianti di ROM che possono essere programmate sul campo e la tecnologia è diventata abbastanza buona da essere quasi sempre utilizzate oggi su ROM maschera. EEPROM, ROM programmabile cancellabile elettricamente, è disponibile in diverse forme, ma è possibile scrivere su tutte. Il tempo di scrittura è in genere molto più lungo del tempo di lettura e il numero di scritture è limitato durante la vita del dispositivo. Una forma particolarmente utile di EEPROM è la memoria flash. La flash viene comunemente utilizzata per archiviare firmware e dati utente che devono essere mantenuti allo spegnimento dell'alimentazione.
La memoria flash, inventata dal Dr. Fujio Masuoka alla Toshiba intorno al 1980, è una forma particolarmente conveniente di memoria non volatile, ma presenta alcune sfide interessanti per i

progettisti di sistemi embedded. In genere, le memorie flash hanno tempi di lettura ragionevolmente veloci, ma non così veloci come SRAM e DRAM, quindi i dati a cui si accede di frequente dovranno in genere essere spostati dalla flash alla RAM prima di essere utilizzati da un programma. I tempi di scrittura sono molto più lunghi dei tempi di lettura e il numero totale di scritture è limitato, quindi queste memorie non sostituiscono la memoria di lavoro.
Esistono due tipi di memorie flash, note come flash NOR e NAND. NOR flash ha tempi di cancellazione e scrittura più lunghi, ma è possibile accedervi come una RAM. La memoria flash NAND è meno costosa e ha tempi di cancellazione e scrittura più rapidi, ma i dati devono essere letti un blocco alla volta, dove un blocco è composto da centinaia a migliaia di bit. Ciò significa che dal punto di vista del sistema si comporta più come un dispositivo di archiviazione secondario come un disco rigido o un supporto ottico come CD o DVD. Entrambi i tipi di flash possono essere cancellati e riscritti solo un numero limitato di volte, in genere meno di 1.000.000 per flash NOR e meno di 10.000.000 per flash NAND, al momento della stesura di questo documento.
I tempi di accesso più lunghi, il numero limitato di scritture e gli accessi a blocchi (per flash NAND), complicano il problema per i progettisti di sistemi embedded. Queste proprietà devono essere prese in considerazione non solo durante la progettazione dell'hardware, ma anche del software.

9.2 Memory Hierarchy

Molte applicazioni richiedono notevoli quantità di memoria, più di quella disponibile su chip in un microcomputer. Molti processori utilizzano una gerarchia di memoria, che combina diverse tecnologie di memoria per aumentare la capacità complessiva della memoria, ottimizzando al contempo costi, latenza e consumo energetico. Tipicamente, una quantità relativamente piccola di SRAM su chip verrà utilizzata con una quantità maggiore di DRAM fuori chip. Questi possono essere ulteriormente combinati con un terzo livello, come le unità disco, che hanno una capacità molto grande, ma mancano di accesso casuale e quindi possono essere piuttosto lente da leggere e scrivere
Il programmatore dell'applicazione potrebbe non essere a conoscenza del fatto che la memoria è frammentata in queste tecnologie. Uno schema comunemente usato chiamato memoria virtuale fa apparire le diverse tecnologie al programmatore come uno spazio di indirizzi contiguo. Il sistema operativo e/o l'hardware fornisce la traduzione degli indirizzi, che converte gli indirizzi logici nello spazio degli indirizzi in posizioni fisiche in una delle tecnologie di memoria disponibili. Questa traduzione è spesso assistita da un componente hardware specializzato chiamato Translation Lookaside Buffer (TLB), che può velocizzare alcune traduzioni di indirizzi. Per un progettista di sistemi embedded, queste tecniche possono creare seri problemi perché rendono molto difficile prevedere o capire quanto tempo impiegheranno gli accessi alla memoria. Pertanto, i progettisti di sistemi embedded in genere devono comprendere il sistema di memoria in modo più approfondito rispetto ai programmatori generici.

9.2.1 Memory Maps

Una mappa di memoria per un processore definisce come gli indirizzi vengono mappati sull'hardware. La dimensione totale dello spazio degli indirizzi è vincolata dalla larghezza dell'indirizzo del processore. Un processore a 32 bit, ad esempio, può indirizzare 2 32 posizioni o 4 gigabyte (GB), supponendo che ogni indirizzo si riferisca a un byte.
Si noti che questa architettura separa gli indirizzi utilizzati per la memoria del programma (etichettata A nella figura) da quelli utilizzati per la memoria dei dati (B e D). Questo modello (tipico) consente l'accesso a

queste memorie tramite bus separati, consentendo il recupero simultaneo di istruzioni e dati. Questo raddoppia efficacemente la larghezza di banda della memoria. Tale separazione della memoria del programma dalla memoria dei dati è nota come architettura di Harvard. Contrasta con la classica architettura von Neumann, che memorizza programma e dati nella stessa memoria.
Qualsiasi realizzazione particolare in silicio di questa architettura è vincolata da questa mappa di memoria. Ad esempio, il controller Luminary Micro1 LM3S8962, che include un core ARM CortexTM - M3, ha 256 KB di memoria flash su chip, per nulla vicino al totale di 0,5 GB consentito dall'architettura. Questa memoria è mappata agli indirizzi da 0x00000000 a 0x0003FFFF. Gli indirizzi rimanenti consentiti dall'architettura per la memoria del programma, che vanno da 0x00040000 a 0x1FFFFFFF, sono "indirizzi riservati", il che significa che non dovrebbero essere utilizzati da un compilatore destinato a questo particolare dispositivo.
L'LM3S8962 ha 64 KB di SRAM, mappati agli indirizzi da 0x20000000 a 0x2000FFFF, una piccola porzione dell'area B nella figura. Include anche una serie di periferiche su chip, che sono dispositivi a cui il processore accede utilizzando alcuni degli indirizzi di memoria nell'intervallo da 0x40000000 a 0x5FFFFFFFF (area C nella figura). Questi includono timer, ADC, GPIO, UART e altri dispositivi I/O. Ciascuno di questi dispositivi occupa alcuni degli indirizzi di memoria fornendo registri mappati in memoria. Il processore può scrivere su alcuni di questi registri per configurare e/o controllare la periferica, o per fornire dati da produrre su un'uscita. Alcuni dei registri possono essere letti per recuperare i dati di input ottenuti dalla periferica. Alcuni degli indirizzi nella regione del bus periferico privato vengono utilizzati per accedere al controller di interrupt.
L'LM3S8962 è montato su un circuito stampato che fornirà dispositivi aggiuntivi come memoria dati DRAM e dispositivi esterni aggiuntivi. Come mostrato nella Figura 9.1, questi verranno mappati su indirizzi di memoria nell'intervallo da 0xA0000000 a 0xDFFFFFFF (area E). Ad esempio, la scheda di valutazione Stellaris R LM3S8962 di Luminary Micro non include memoria esterna aggiuntiva, ma aggiunge alcuni dispositivi esterni come un display LCD, uno slot MicroSD per memoria flash aggiuntiva e un'interfaccia USB.
Ciò lascia molti indirizzi di memoria inutilizzati. ARM ha introdotto un modo intelligente per sfruttare questi indirizzi inutilizzati chiamato bit banding, in cui alcuni degli indirizzi inutilizzati possono essere utilizzati per accedere a singoli bit anziché a interi byte o parole nella memoria e nelle periferiche. Ciò rende alcune operazioni più efficienti, poiché le istruzioni aggiuntive per mascherare i bit desiderati diventano superflue.

Harvard Architecture

Il termine "architettura di Harvard" deriva dal computer Mark I, che utilizzava memorie distinte per programma e dati. Il Mark I è stato realizzato con relè elettromeccanici da IBM e spedito ad Harvard nel 1944. La macchina memorizzava istruzioni su nastro perforato e dati in contatori elettromeccanici. È stato chiamato Automatic Sequence Controlled Calculator (ASCC) da IBM ed è stato ideato da Howard H. Aiken per risolvere numericamente equazioni differenziali. Il contrammiraglio Grace Murray Hopper della Marina degli Stati Uniti e il finanziamento dell'IBM sono stati determinanti nel trasformare la macchina in realtà.

9.2.2 Register Files

La memoria più strettamente integrata in un processore è il file di registro. Ciascun registro nel file memorizza una parola. La dimensione di una parola è una proprietà chiave dell'architettura di un processore. È un byte su un'architettura a 8 bit, quattro byte su un'architettura a 32 bit e otto byte su un'architettura a 64 bit. Il file di registro può essere implementato direttamente utilizzando i flip flop nel circuito del processore, oppure i registri possono essere raccolti in un unico banco di memoria, tipicamente utilizzando la stessa tecnologia SRAM discussa sopra. Il numero di registri in un processore è generalmente

piccolo. La ragione di ciò non è tanto il costo dell'hardware del file di registro, quanto piuttosto il costo dei bit in una parola di istruzione. Un'architettura del set di istruzioni (ISA) fornisce in genere istruzioni che possono accedere a uno, due o tre registri. Per memorizzare in modo efficiente i programmi in memoria, queste istruzioni non possono richiedere troppi bit per codificarli e quindi non possono dedicare troppi bit all'identificazione dei registri. Se il file di registro ha 16 registri, ogni riferimento a un registro richiede 4 bit. Se un'istruzione può fare riferimento a 3 registri, ciò richiede un totale di 12 bit. Se una parola di istruzione è, ad esempio, di 16 bit, rimangono solo 4 bit per altre informazioni nell'istruzione, come l'identità dell'istruzione stessa, che deve anche essere codificata nell'istruzione. Questo identifica, ad esempio, se l'istruzione specifica che devono essere aggiunti o sottratti due registri, con il risultato memorizzato nel terzo registro.

9.2.3 Scratchpads and Caches

Molte applicazioni integrate combinano tecnologie di memoria. Si accede ad alcuni ricordi prima di altri; diciamo che i primi sono “più vicini” al processore rispetto ai secondi. Ad esempio, una memoria chiusa (SRAM) viene in genere utilizzata per archiviare temporaneamente i dati di lavoro mentre il programma opera su di essa. Se la memoria chiusa ha un insieme distinto di indirizzi e il programma è responsabile dello spostamento dei dati al suo interno o all'esterno nella memoria lontana, viene chiamato scratchpad. Se la memoria di chiusura duplica i dati nella memoria lontana con l'hardware che gestisce automaticamente la copia da e verso, viene chiamata cache. Per le applicazioni embedded con stretti vincoli di tempo reale, le memorie cache presentano alcuni formidabili ostacoli perché il loro comportamento temporale può variare sostanzialmente in modi difficili da prevedere. D'altra parte, la gestione manuale dei dati in una memoria scratchpad può essere piuttosto noiosa per un programmatore e i metodi automatici guidati dal compilatore per farlo sono agli inizi.
Come spiegato nella Sezione 9.2.1, un'architettura supporterà in genere uno spazio di indirizzi molto più ampio di quello che può essere effettivamente memorizzato nella memoria fisica del processore, con un sistema di memoria virtuale utilizzato per presentare al programmatore la vista di uno spazio di indirizzi contiguo . Se il processore è dotato di un'unità di gestione della memoria (MMU), i programmi fanno riferimento a indirizzi logici e la MMU li traduce in indirizzi fisici.
Pertanto, al programma viene data l'illusione di una grande quantità di memoria, con il costo che i tempi di accesso alla memoria diventano piuttosto difficili da prevedere. Non è raro che i tempi di accesso alla memoria varino di un fattore pari o superiore a 1000, a seconda di come gli indirizzi logici vengono distribuiti nelle memorie fisiche.

9.3 Memory Models

Un modello di memoria definisce come la memoria viene utilizzata dai programmi. L'hardware, il sistema operativo (se presente) e il linguaggio di programmazione e il relativo compilatore contribuiscono tutti al modello di memoria. Questa sezione discute alcuni dei problemi comuni che sorgono con i modelli di memoria.

9.3.1 Memory Addresses

Come minimo, un modello di memoria definisce un intervallo di indirizzi di memoria accessibili al programma. In C, questi indirizzi sono memorizzati in puntatori. In un'architettura a 32 bit, gli indirizzi di memoria sono numeri interi senza segno a 32 bit, in grado di rappresentare gli indirizzi da 0 a 2 32 − 1, ovvero circa quattro miliardi di indirizzi.

9.3.2 Stacks

Uno stack è una regione di memoria allocata dinamicamente al programma in un modello LIFO (last-in, first-out). Un puntatore dello stack (in genere un registro) contiene l'indirizzo di memoria della parte superiore dello stack. Quando un elemento viene inserito nella pila, il puntatore della pila viene incrementato e l'elemento viene archiviato nella nuova posizione a cui fa riferimento il puntatore della pila. Quando un elemento viene estratto dallo stack, la posizione di memoria a cui fa riferimento il puntatore dello stack viene (in genere) copiata da qualche altra parte (ad esempio, in un registro) e il puntatore dello stack viene decrementato.
Gli stack vengono in genere utilizzati per implementare le chiamate di procedura.
I dati per una procedura che viene inserita nello stack sono noti come stack frame di quella procedura. Quando una procedura ritorna, il compilatore apre il suo stack frame, recuperando finalmente la posizione del programma in cui riprendere l'esecuzione.
Per il software incorporato, può essere disastroso se il puntatore dello stack viene incrementato oltre la memoria allocata per lo stack. Un tale overflow dello stack può comportare la sovrascrittura della memoria utilizzata per altri scopi, portando a risultati imprevedibili.

9.3.3 Memory Protection Units

Un problema chiave nei sistemi che supportano più attività simultanee è impedire a un'attività di interrompere l'esecuzione di un'altra. Ciò è particolarmente importante nelle applicazioni integrate che consentono il download di software di terze parti, ma può anche fornire un'importante difesa contro i bug del software nelle applicazioni critiche per la sicurezza.

9.3.4 Dynamic Memory Allocation

Le applicazioni software generiche hanno spesso requisiti indeterminati di memoria, a seconda dei parametri e/o dell'input dell'utente. Per supportare tali applicazioni, gli informatici hanno sviluppato schemi di allocazione di memoria dinamica, in cui un programma può richiedere in qualsiasi momento che il sistema operativo allochi memoria aggiuntiva. La memoria viene allocata da una struttura di dati nota come heap, che facilita il monitoraggio di quali porzioni di memoria sono utilizzate da quale applicazione.
Il supporto per l'allocazione della memoria spesso (ma non sempre) include il Garbage Collection. Ad esempio, la raccolta dei rifiuti è intrinseca nel linguaggio di programmazione Java. Un Garbage Collector è un'attività che viene eseguita periodicamente o quando la memoria si esaurisce che analizza le strutture di dati che un programma ha allocato e libera automaticamente tutte le porzioni di memoria a cui non viene più fatto riferimento all'interno del programma. Quando si utilizza un Garbage Collector, in linea di principio, un programmatore non deve preoccuparsi di liberare esplicitamente memoria.

Con o senza Garbage Collection, è possibile che un programma accumuli inavvertitamente memoria che non viene mai liberata.

10. Input and Output

Poiché i sistemi cyber-fisici integrano l'elaborazione e la dinamica fisica, i meccanismi nei processori che supportano l'interazione con il mondo esterno sono centrali in qualsiasi progetto. Un progettista di sistema deve affrontare una serie di problemi.

10.1 I/O Hardware

I processori incorporati, siano essi microcontrollori, processori DSP o processori generici, in genere includono una serie di meccanismi di input e output (I/O) su chip, esposti ai progettisti come pin del chip. La scheda di valutazione nell'esempio sopra è più di un processore poiché include un display e varie interfacce hardware (interruttori e un altoparlante, per esempio). Tale scheda è spesso chiamata computer a scheda singola o scheda per microcomputer.

10.1.1 Pulse Width Modulation

La modulazione della larghezza di impulso (PWM) è una tecnica per fornire una quantità variabile di potenza in modo efficiente a dispositivi hardware esterni. Può essere utilizzato per controllare ad esempio la velocità di motori elettrici, la luminosità di una luce LED e la temperatura di un elemento riscaldante. In generale, può fornire quantità variabili di alimentazione a dispositivi che tollerano variazioni rapide e brusche di tensione e corrente.
L'hardware PWM utilizza solo circuiti digitali e quindi è facile da integrare sullo stesso chip con un microcontrollore. I circuiti digitali, in base alla progettazione, producono solo due livelli di tensione, alto e basso. Un segnale PWM passa rapidamente da alto a basso a una frequenza fissa, variando la quantità di tempo in cui mantiene alto il segnale. Il duty cycle è la proporzione di tempo in cui la tensione è alta. Se il duty cycle è del 100%, la tensione è sempre alta. Se il duty cycle è 0%, la tensione è sempre bassa.
Molti microcontrollori forniscono dispositivi periferici PWM (vedi Figura 10.1). Per utilizzarli, un programmatore in genere scrive un valore in un registro mappato in memoria per impostare il ciclo di lavoro (la frequenza può anche essere impostabile). Il dispositivo fornisce quindi alimentazione all'hardware esterno in proporzione al ciclo di lavoro specificato.
PWM è un modo efficace per fornire quantità variabili di energia, ma solo a determinati dispositivi

10.1.2 General-Purpose Digital I/O

I progettisti di sistemi embedded hanno spesso bisogno di collegare hardware digitale specializzato o personalizzato a processori embedded. Molti processori embedded hanno una serie di pin I/O generici (GPIO), che consentono al software di leggere o scrivere livelli di tensione che rappresentano uno zero o

uno logico.
In molti modelli, un pin GPIO può essere configurato per essere un'uscita. Ciò consente al software di scrivere su un registro mappato in memoria per impostare la tensione di uscita su alta o bassa. Con questo meccanismo, il software può controllare direttamente i dispositivi fisici esterni
Tuttavia, la cautela è d'obbligo. Quando si interfaccia l'hardware ai pin GPIO, un progettista deve comprendere le specifiche del dispositivo. In particolare, i livelli di tensione e corrente variano a seconda del dispositivo. Se un pin GPIO produce una tensione di uscita di VDD quando ne viene fornita una logica, il progettista deve conoscere i limiti di corrente prima di collegare un dispositivo ad esso
È essenziale mantenere questa corrente entro le tolleranze specificate. Il superamento di queste tolleranze potrebbe causare il surriscaldamento e il guasto del dispositivo. Potrebbe essere necessario un amplificatore di potenza per fornire una corrente adeguata. Potrebbe essere necessario anche un amplificatore per modificare i livelli di tensione.
Inoltre, può essere importante mantenere l'isolamento elettrico tra i circuiti del processore e i dispositivi esterni.
Una strategia utile consiste nel dividere un circuito in domini elettrici, possibilmente con alimentatori separati, che hanno un'influenza relativamente scarsa l'uno sull'altro.
I pin GPIO possono anche essere configurati come ingressi, nel qual caso il software sarà in grado di reagire ai livelli di tensione forniti dall'esterno.
I pin GPIO possono anche essere configurati come ingressi, nel qual caso il software sarà in grado di reagire ai livelli di tensione forniti dall'esterno.
In molte applicazioni, più dispositivi possono condividere un'unica connessione elettrica.

10.1.3 Serial Interfaces

Uno dei principali vincoli affrontati dai progettisti di processori embedded è la necessità di avere contenitori fisicamente piccoli e un basso consumo energetico. Una conseguenza è che il numero di pin sul circuito integrato del processore è limitato. Pertanto, ogni pin deve essere utilizzato in modo efficiente. Inoltre, quando si cablano insieme i sottosistemi, il numero di cavi deve essere limitato per tenere sotto controllo l'ingombro e il costo complessivi del prodotto. Pertanto, anche i cavi devono essere utilizzati in modo efficiente. Un modo per utilizzare pin e fili in modo efficiente consiste nell'inviare informazioni su di essi in serie come sequenze di bit. Tale interfaccia è chiamata interfaccia seriale. Un certo numero di standard si è evoluto per le interfacce seriali in modo che i dispositivi di diversi produttori possano (di solito) essere collegati.
Un microcontrollore utilizzerà in genere un ricevitore/trasmettitore asincrono universale (UART) per convertire il contenuto di un registro a 8 bit in una sequenza di bit per la trasmissione su un collegamento seriale RS-232.
I dispositivi più recenti progettati per la connessione ai personal computer utilizzano in genere interfacce USB (Universal Serial Bus), standardizzate da un consorzio di fornitori.
USB è elettricamente più semplice di RS-232 e utilizza connettori più semplici e robusti, come mostrato nella Figura 10.4. Ma lo standard USB definisce molto di più del trasporto elettrico di byte e per supportarlo è necessaria una logica di controllo più complicata

Ci sono molte altre interfacce seriali in uso oggi, tra cui ad esempio I 2C (circuito interintegrato), SPI (bus dell'interfaccia periferica seriale), PCI Express (peripheral component interconnect express), FireWire, MIDI (interfaccia digitale per strumenti musicali) e versioni seriali di SCSI (descritto di seguito).

10.1.4 Parallel Interfaces

Un'interfaccia seriale invia o riceve una sequenza di bit in sequenza su una singola riga. Un'interfaccia parallela utilizza più linee per inviare bit contemporaneamente. Naturalmente, ogni linea di un'interfaccia parallela è anche un'interfaccia seriale, ma il raggruppamento logico e l'azione coordinata di queste linee è ciò che rende l'interfaccia un'interfaccia parallela.
Sembra intuitivo che le interfacce parallele debbano fornire prestazioni più elevate rispetto alle interfacce seriali, poiché per l'interconnessione vengono utilizzati più cavi. Tuttavia, questo non è necessariamente il caso. Una sfida significativa con le interfacce parallele è mantenere la sincronia tra più fili. Ciò diventa più difficile all'aumentare della lunghezza fisica dell'interconnessione. Questo fatto, combinato con la necessità di cavi più ingombranti e più pin I/O, ha portato alla sostituzione di molte interfacce tradizionalmente parallele con interfacce seriali.

10.1.5 Buses

Un bus è un'interfaccia condivisa tra più dispositivi, a differenza di un'interconnessione punto a punto che collega esattamente due dispositivi. I bus possono essere interfacce seriali (come USB) o interfacce parallele.
Poiché un bus è condiviso tra più dispositivi, qualsiasi architettura bus deve includere un protocollo MAC (Media Access Control) per arbitrare gli accessi concorrenti. Un semplice protocollo MAC ha un unico bus master che interroga gli slave bus.

10.2.1 Interrupts and Exceptions

Un interrupt è un meccanismo per sospendere l'esecuzione di qualsiasi cosa un processore stia attualmente facendo ed eseguire una sequenza di codice predefinita chiamata routine di servizio di interrupt (ISR) o gestore di interrupt. Tre tipi di eventi possono attivare un'interruzione. Uno è un interrupt hardware, in cui alcuni hardware esterni cambiano il livello di tensione su una linea di richiesta di interrupt. Nel caso di un'interruzione software, il programma in esecuzione attiva l'interruzione eseguendo un'istruzione speciale o scrivendo in un registro mappato in memoria. Una terza variante è denominata eccezione, in cui l'interruzione viene attivata dall'hardware interno che rileva un errore, ad esempio un errore di segmentazione.
Al verificarsi di un trigger di interrupt, l'hardware deve prima decidere se rispondere. Se gli interrupt sono disabilitati, non risponderà. Il meccanismo per abilitare o disabilitare gli interrupt varia in base al processore. Inoltre, è possibile che alcuni interrupt siano abilitati e altri no. Gli interrupt e le eccezioni generalmente hanno priorità e un interrupt verrà servito solo se il processore non è già nel mezzo della manutenzione di un interrupt con una priorità più alta. In genere, le eccezioni hanno la priorità più alta e sono sempre gestite.