

11. Structured design for real-time systems

- Guidance for code structuring
- Data definitions
- Identification of tasks
- Task sequence model with initial priority settings
- Inter-task dependencies and communication needs
- Highlight critical sections
- Reveal potential deadlock situations
- Guide the production of test data
- Offer flexible/adaptable implementations
- Assistance during debugging

Fig. 11.2a Requirements for a r-t design method

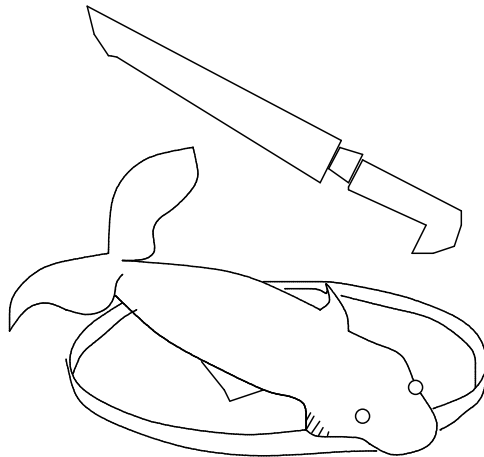


Fig. 11.2b The designer's dilemma: where to make the first cut

- Unambiguous nomenclature for expressing ideas and relationships
- Disciplined sets of questions
- Clear timetable of activities
- Language for communicating with colleagues and clients
- Support for the identification of core data structures
- Guidelines for problem decomposition

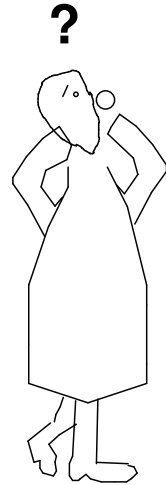
Fig. 11.2c Initial project needs

- Clear route for implementation
- Code documentation for reference for debugging
- Trial data for acceptance tests
- Documentation for customer's records
- Support for continuing maintenance activity

Fig. 11.2d Subsequent project needs

- Decisions too early
- Problem segmentation
- Semantic gap
- Code repetition
- Critical problems

Fig. 11.3 Design issues



- Communication with others
- Abstraction of relevant aspects
- Intellectual discipline
- Temporal -> Spatial mapping
- Record of decisions
- Unambiguous representation

Fig. 11.4 Reasons for using diagrams

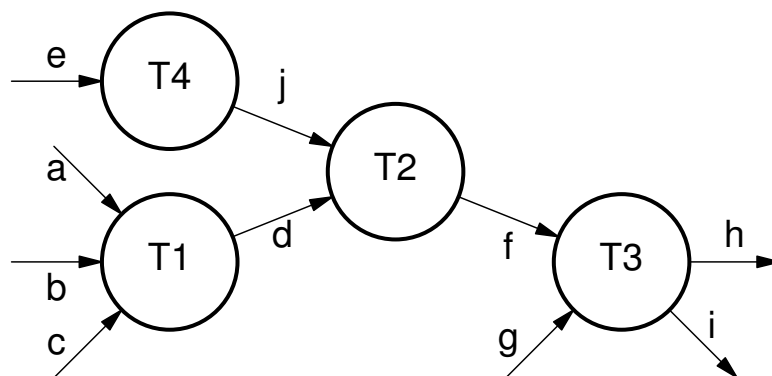


Fig. 11.5a Data Flow Diagram (DFD)

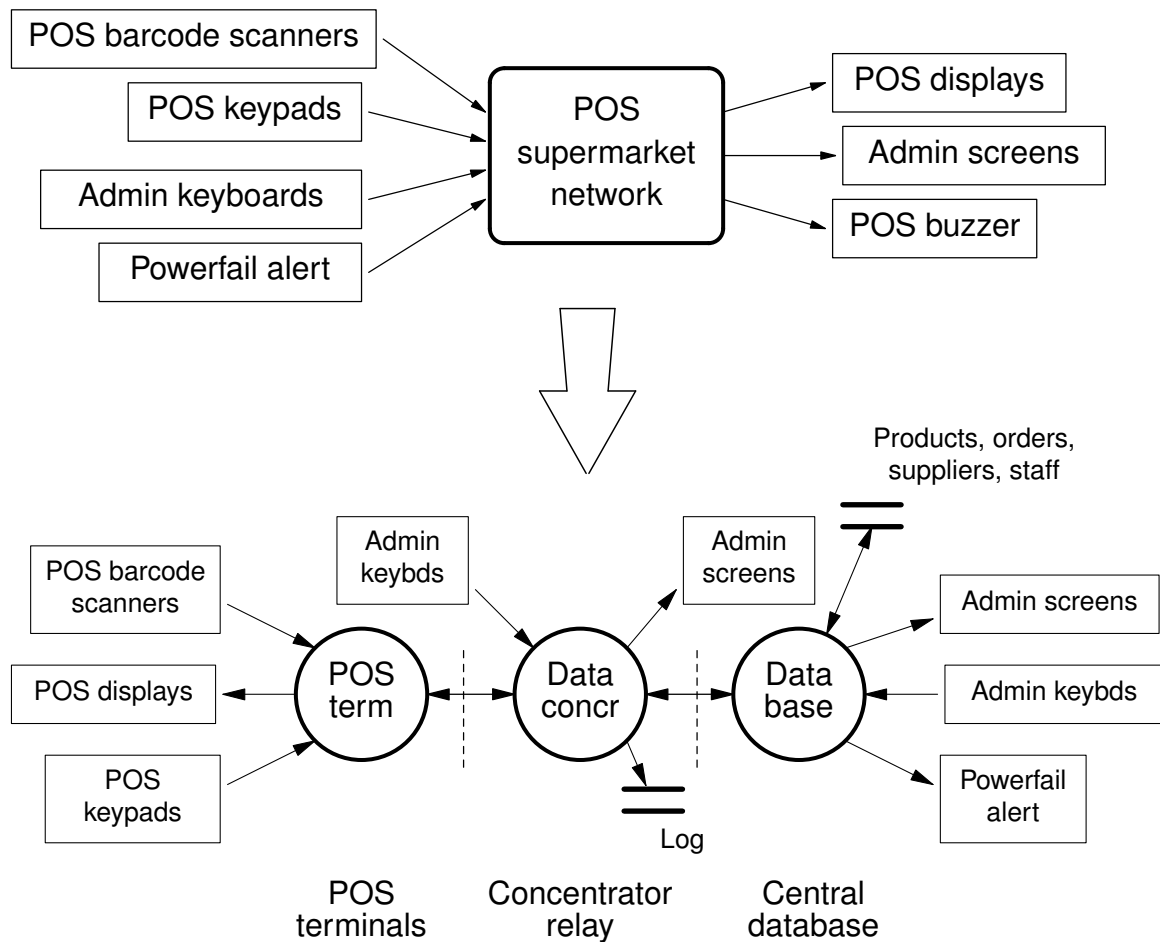


Fig. 11.5b Further partitioning of a context diagram into level-I DFD

```

while (1) {
    a = read();
    b = read();
    c = read();
    d = doT1(a, b, c);
    e = read();
    j = doT4(e);
    f = doT2(j, d);
    g = read();
    doT3(f, g, &h, &i);
    write(h);
    write(i);
}

```

Fig. 11.6a Direct coding of DFD (from Fig. 11.5a)

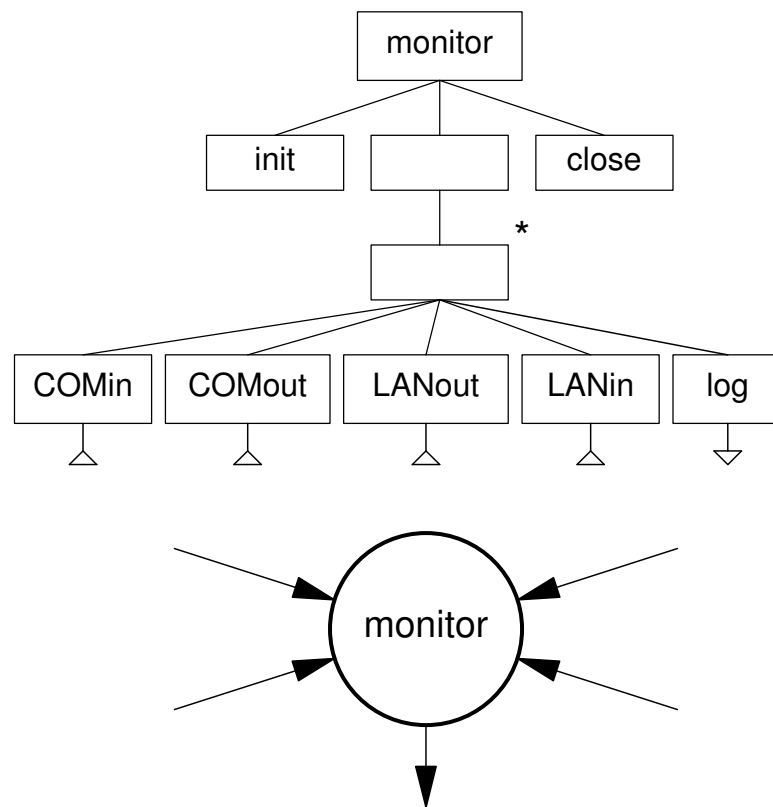


Fig. 11.6b Structure chart for a single data process (from Fig. 11.5b)

		Rules								
Conditions		1	2	3	4	5	6	7	8	9
Destn	to me	1		1		1	1	1		
	to other		1		1				1	1
Source	from me	1		1		1			1	1
	from other		1		1		1	1		
Type	login	1	1							
	logout			1	1					
	data					1	1		1	
	ACK							1		1
	NAK									
Actions										
	pass on		1		1					
	delete	1		1		1		1		
	display					1	1			
	send ACK		1				1			
	send NAK									
	logon	1								
	logoff			1						
	update		1	1	1	1				
	re-tx									
	1	1								

Fig. 11.6c Decision table for decoding a store-and-forward LAN packet

To	From	Type	Comment	
me	me	login	successful login request, init directory	00000
		logout	successful logout request, clear directory	00001
		data	test messg, display, return ACK	00010
		ACK	for earlier test messg sent, del from pending tbl	00011
		NAK	for earlier test messg sent, resend from pending tbl	00100
	other	login	error	01000
		logout	error	01001
		data	messg arrives, display, return ACK	01010
		ACK	for earlier messg sent, del from pending tbl	01011
		NAK	for earlier messg sent, resend from pending tbl	01100
other	me	login	error	10000
		logout	error	10001
		data	returned packet, destination problem, resend	10010
		ACK	returned packet, destination problem, del, display	10011
		NAK	returned packet, destination problem, del, display	10100
	other	login	pass packet onward, ACK, update directory	11000
		logout	pass packet onward, update directory	11001
		data	pass packet onward	11010
		ACK	pass packet onward, update directory	11011
		NAK	pass packet onward	11100

Fig. 11.6d Alternate format for decision table from Fig. 11.6c

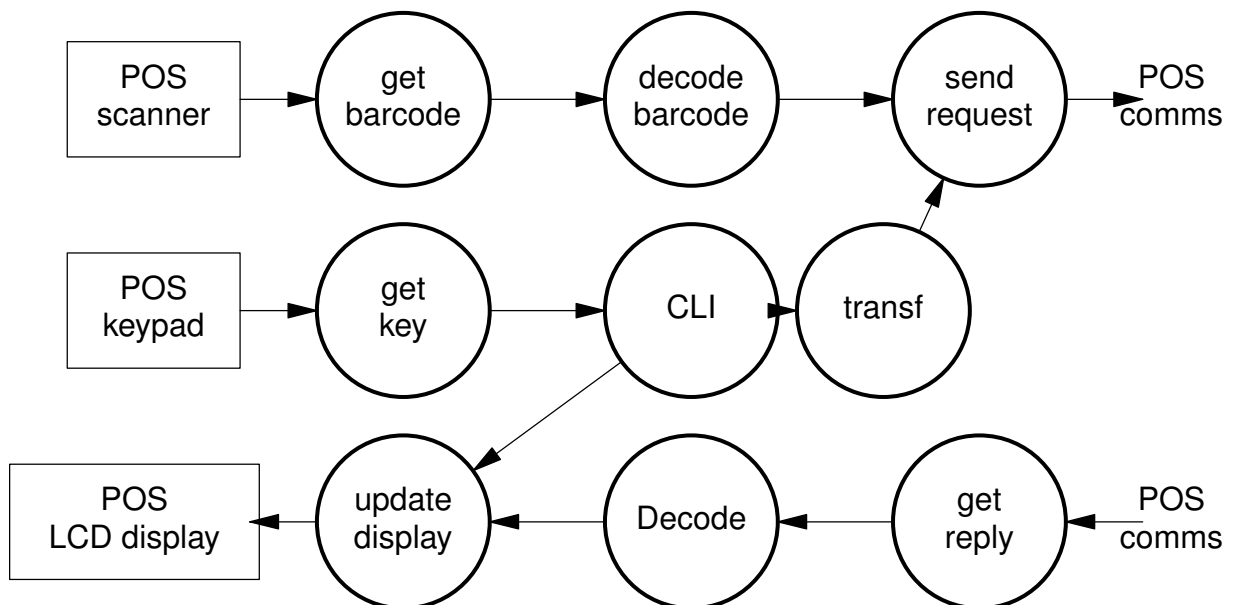


Fig. 11.6e DFD for one of the POS terminals

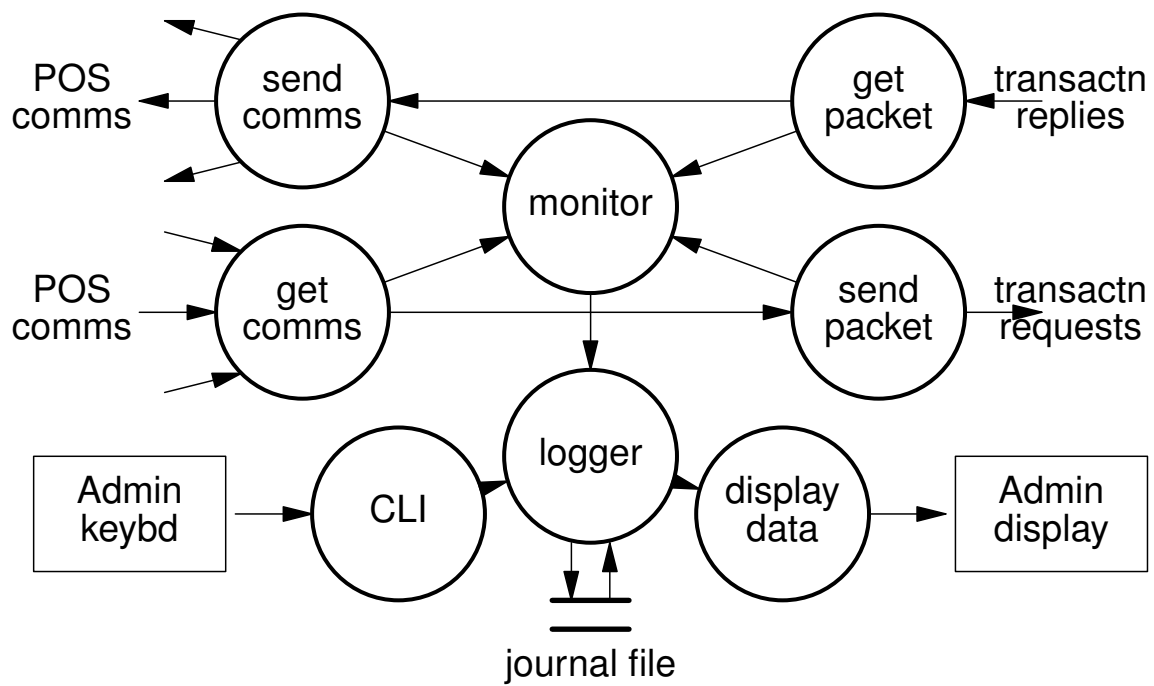


Fig. 11.6f DFD for the Data Concentrator

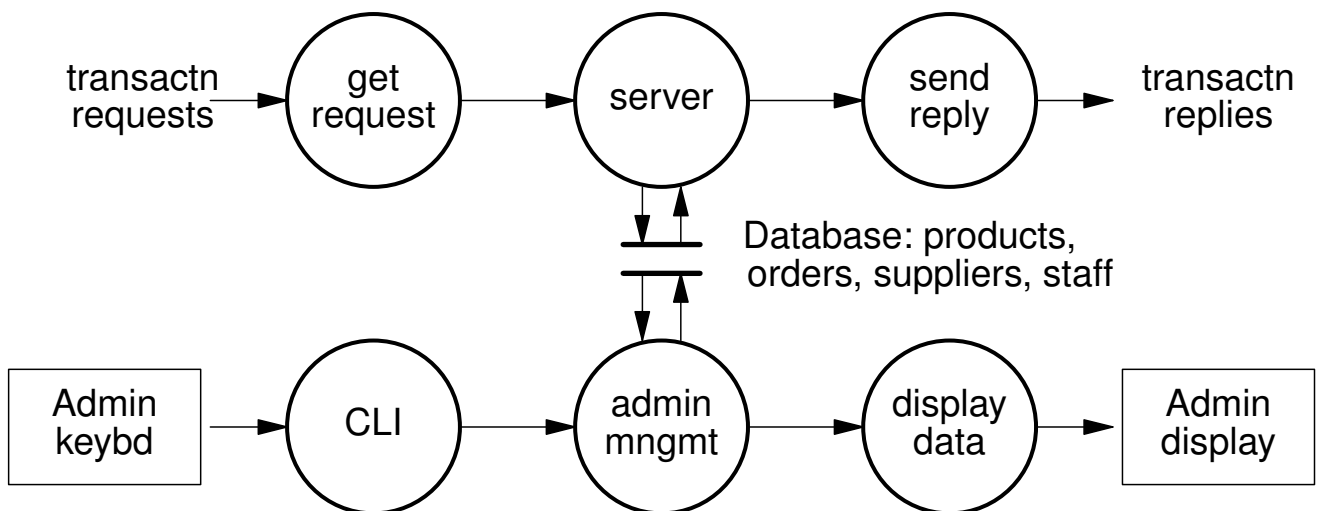


Fig. 11.6g DFD for the database server

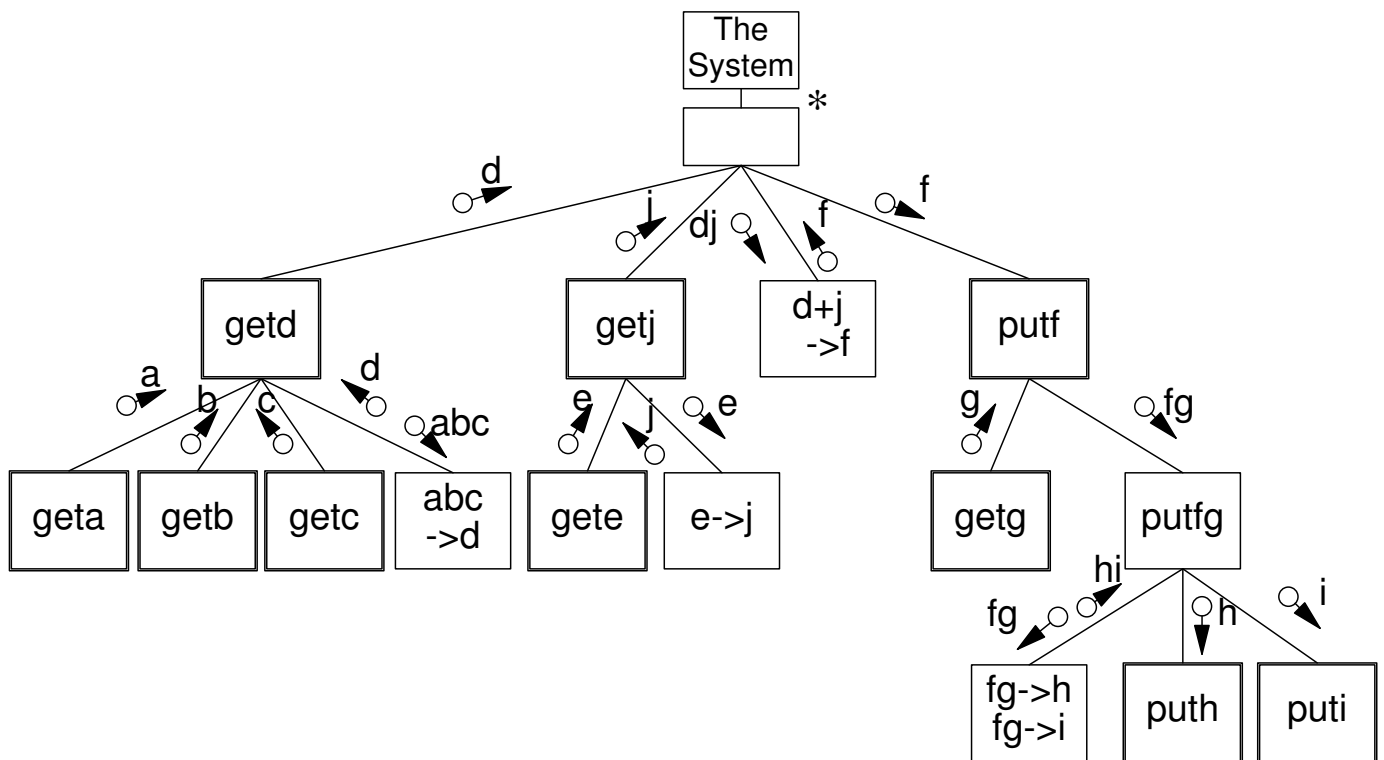


Fig. 11.6h Structure chart derived from the DFD (from Fig. 11.5a)

```

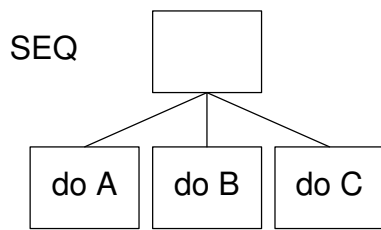
atype geta(){
    atype x;
    x = read();
    return x;
}
btype getb(){
    btype x;
    x = read();
    return x;
}
ctype getc(){
    ctype x;
    x = read();
    return x;
}
dtype convabc(atype w, btype, ctype y){
    dtype z;
    z = f(w, x, y);
    return z;
}
dtype getd(){
    atype w; btype x; ctype y; dtype z;
    w = geta();
    x = getb();
    y = getc();
    z = convabc(w, x, y);
    return z;
}
etype gete(){
    etype x;
    x = read();
    return x;
}
jtype conve(etype x){
    jtype y;
    y = f(x);
    return y;
}

jtype getj(){
    jtype y;
    e = gete();
    y = conve(e);
    return y;
}
ftype convdj(dtype x, jtype y){
    ftype z;
    z = f(d, j);
    return z;
}
gtype getg(){
    gtype x;
    x = read();
    return x;
}
void puth(htype x){
    write(x);
}
void puti(itype x){
    write(x);
}
void putfg(ftype w, gtype x){
    htype y; itype z;
    convfg(w, x, &y, &z);
    puth(y);
    puti(z);
}
void putf(ftype x){
    gtype y;
    y = getg();
    putfg(x, y);
}

main() {
    dtype d; jtype j; ftype f;
    while (1) {
        d = getd();
        j = getj();
        f = convdj(d, j);
        putf(f);
    }
}

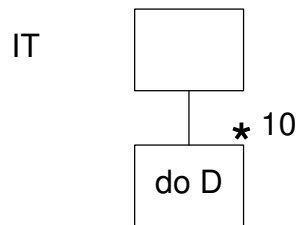
```

Fig. 11.6i Structured coding of DFDs, (from Fig. 11.5a)



```

void doitall(void) {
    doA( );
    doB( );
    doC( );
}
  
```



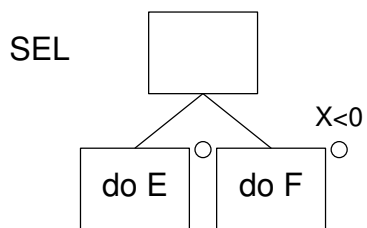
```

for(i=0; i<10; i++) {
    doD( );
}
  
```

or

```

x = 0;
while (x < 10) {
    doD( );
    x++;
}
  
```



```

if (x >= 0) {
    doE( );
} else {
    doF( );
}
  
```

Fig. 11.7a Structure chart representation of SEQ, IT & SEL

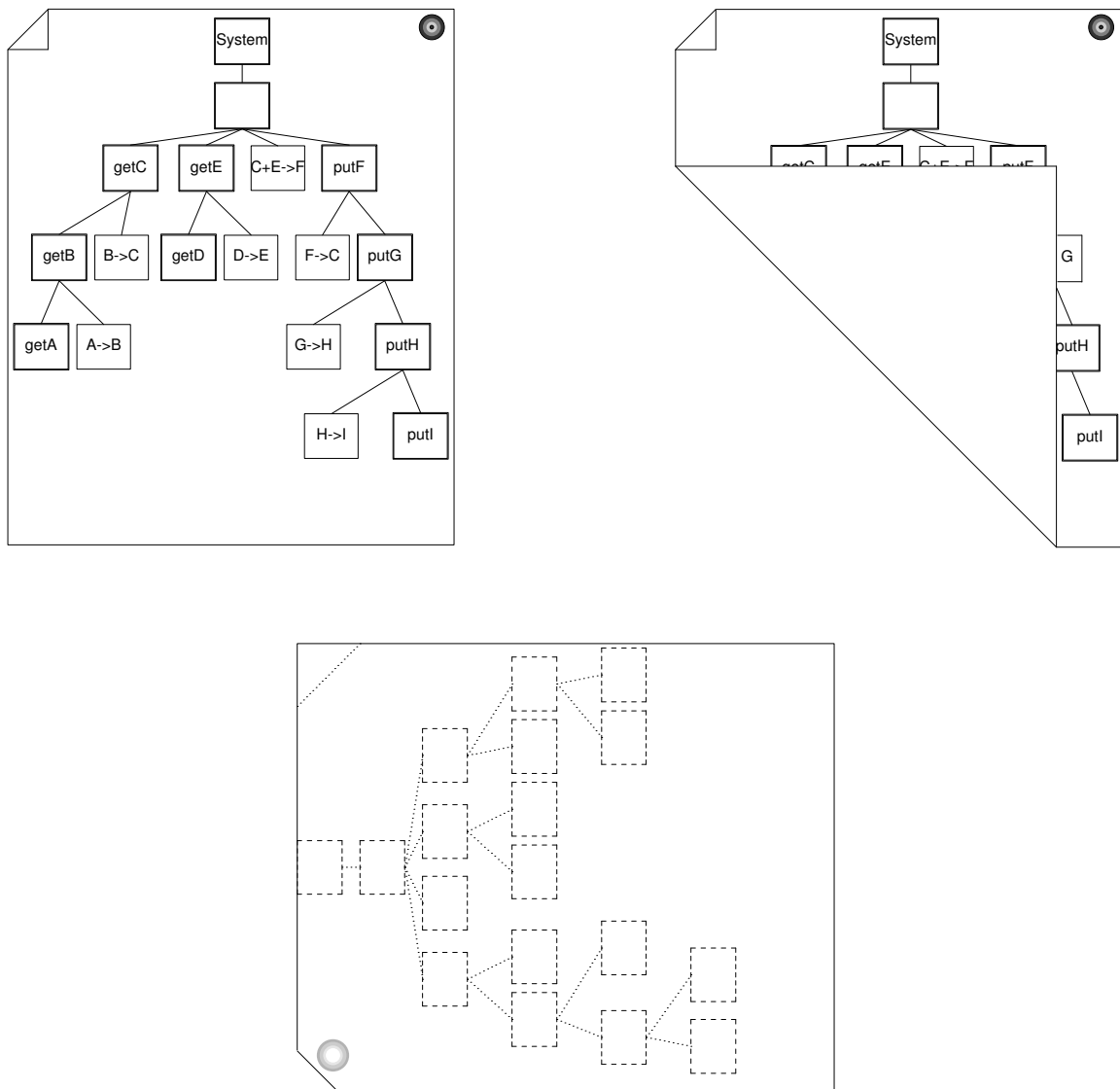


Fig. 11.7b Reading a structure chart for code ordering

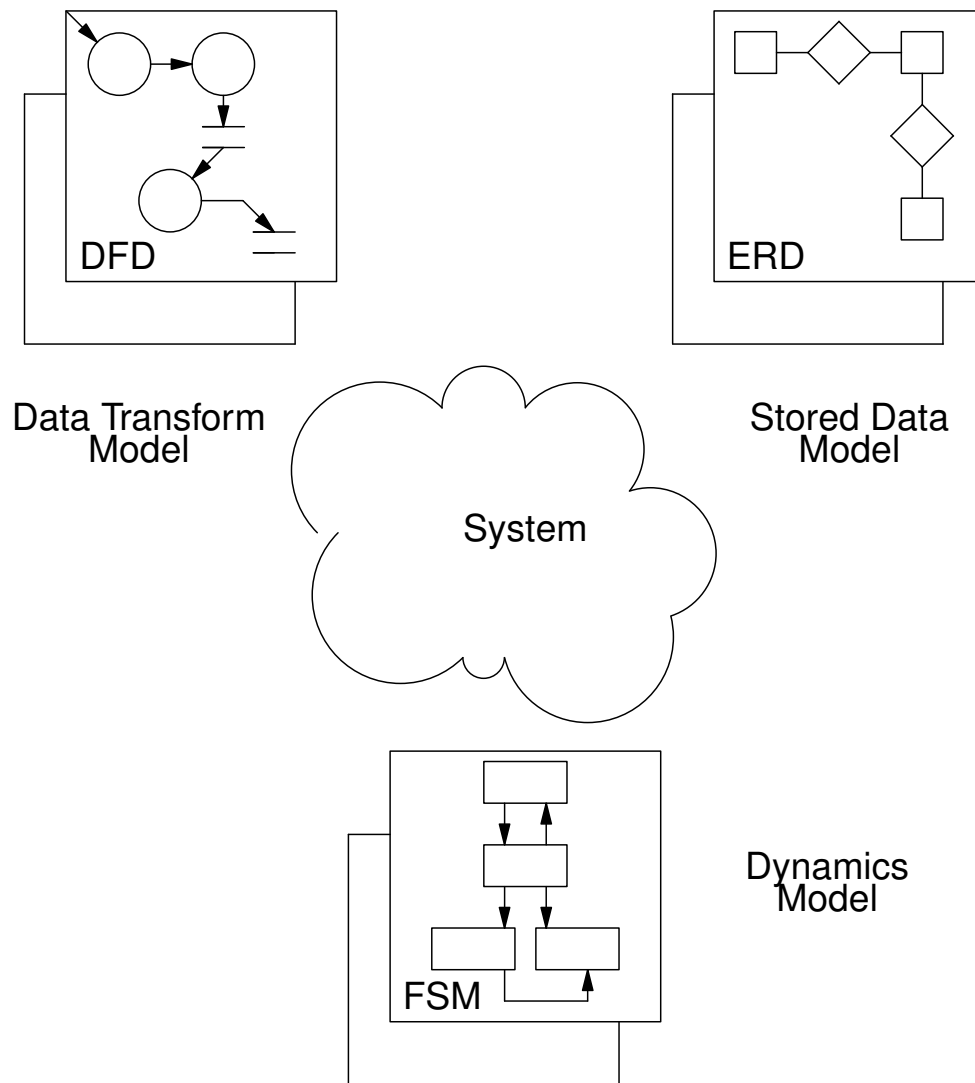


Fig. 11.8 The 3 Yourdon modelling perspectives

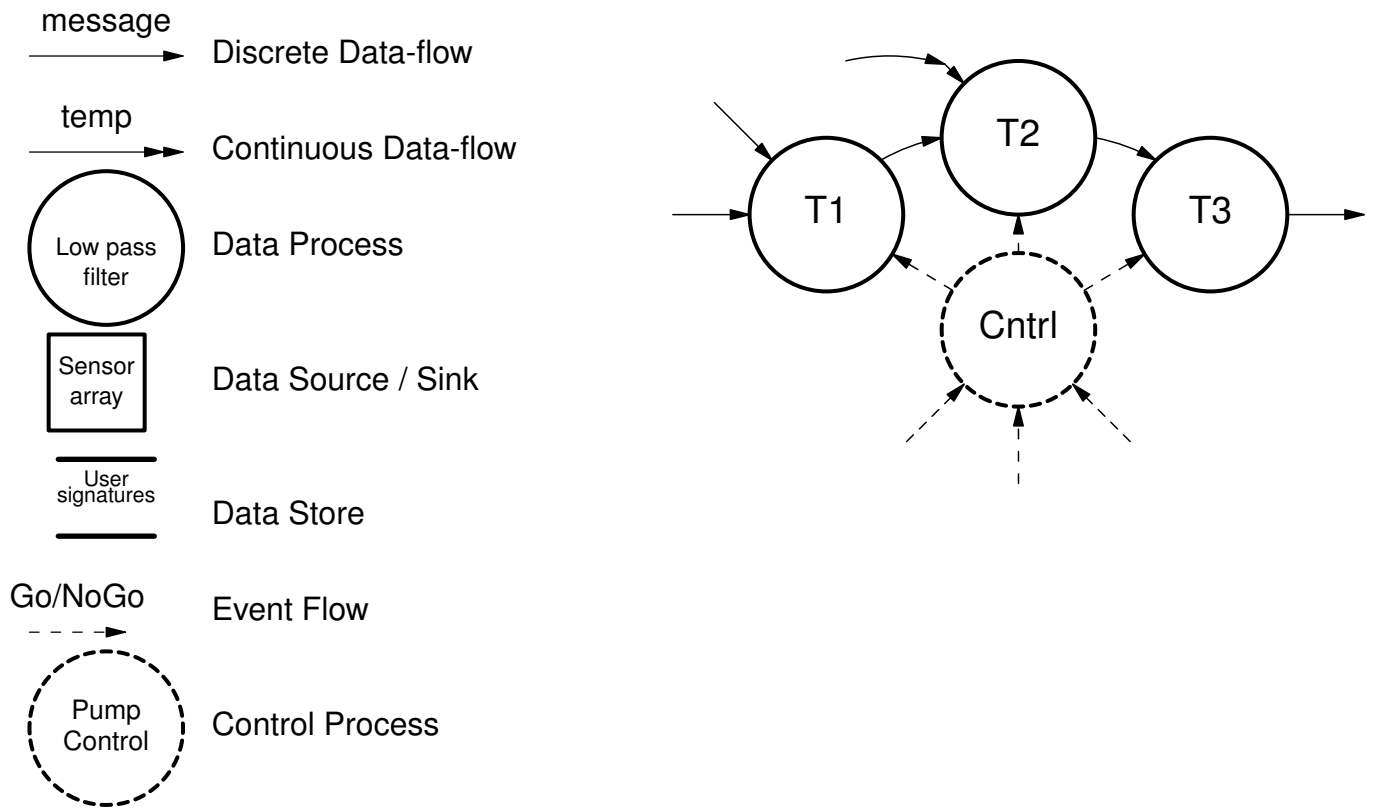


Fig. 11.9a DFD with Control Process

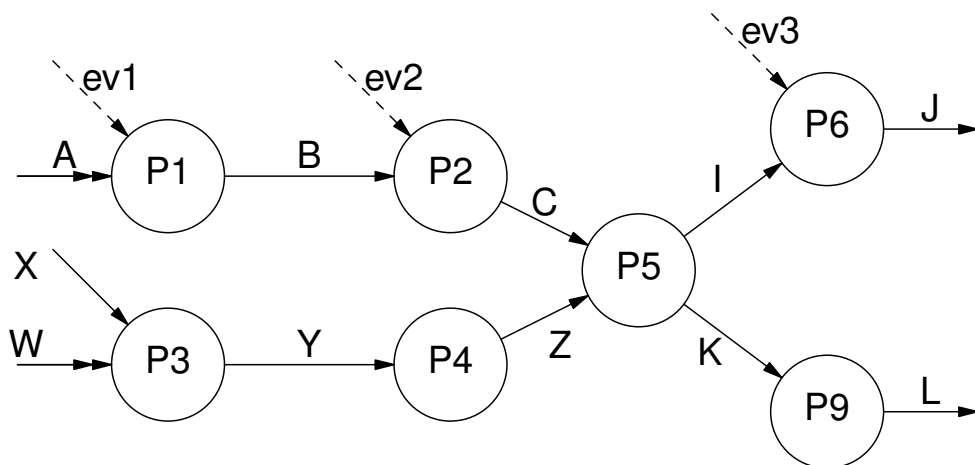


Fig. 11.9b DFD with events

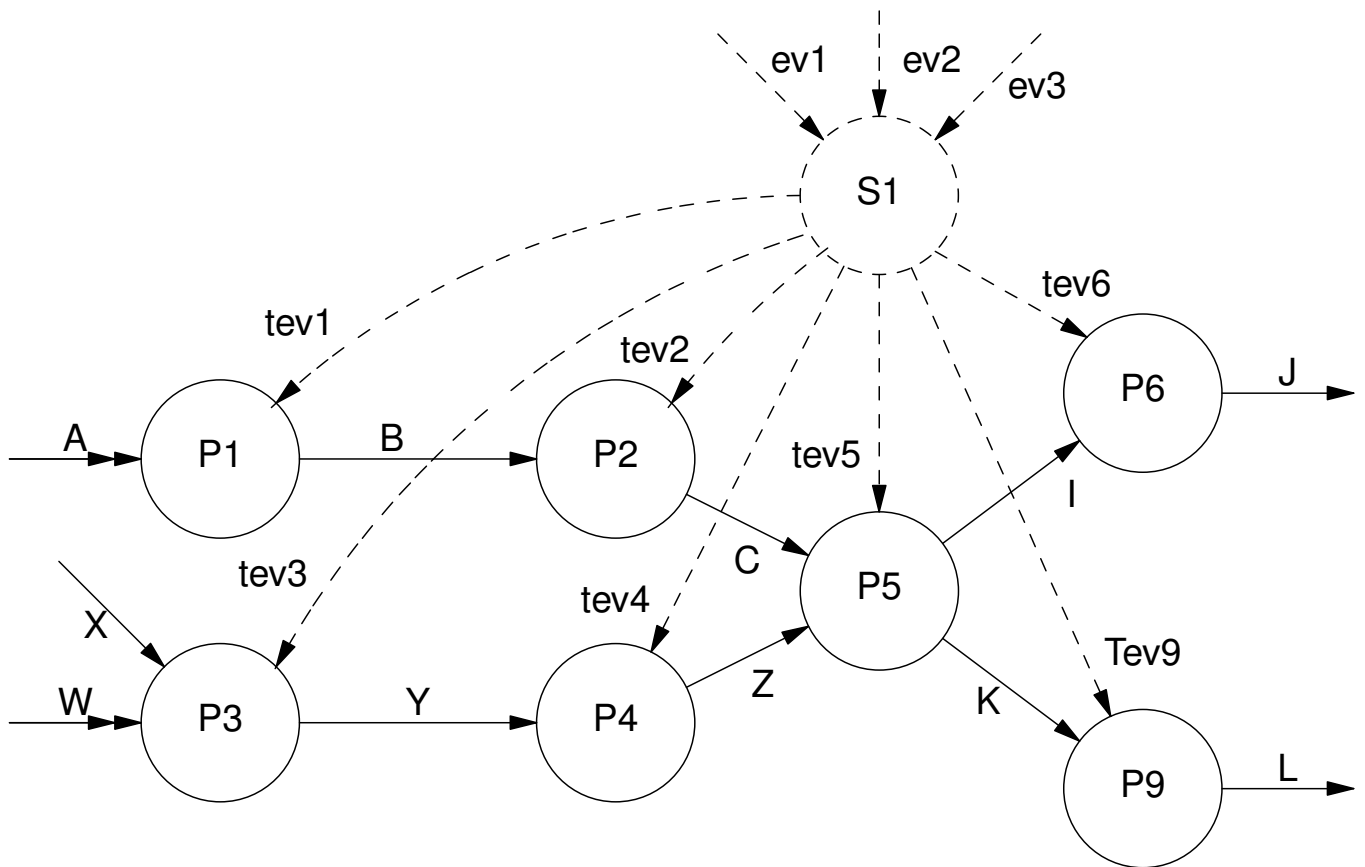


Fig. 11.9c DFD with Control Process Inserted

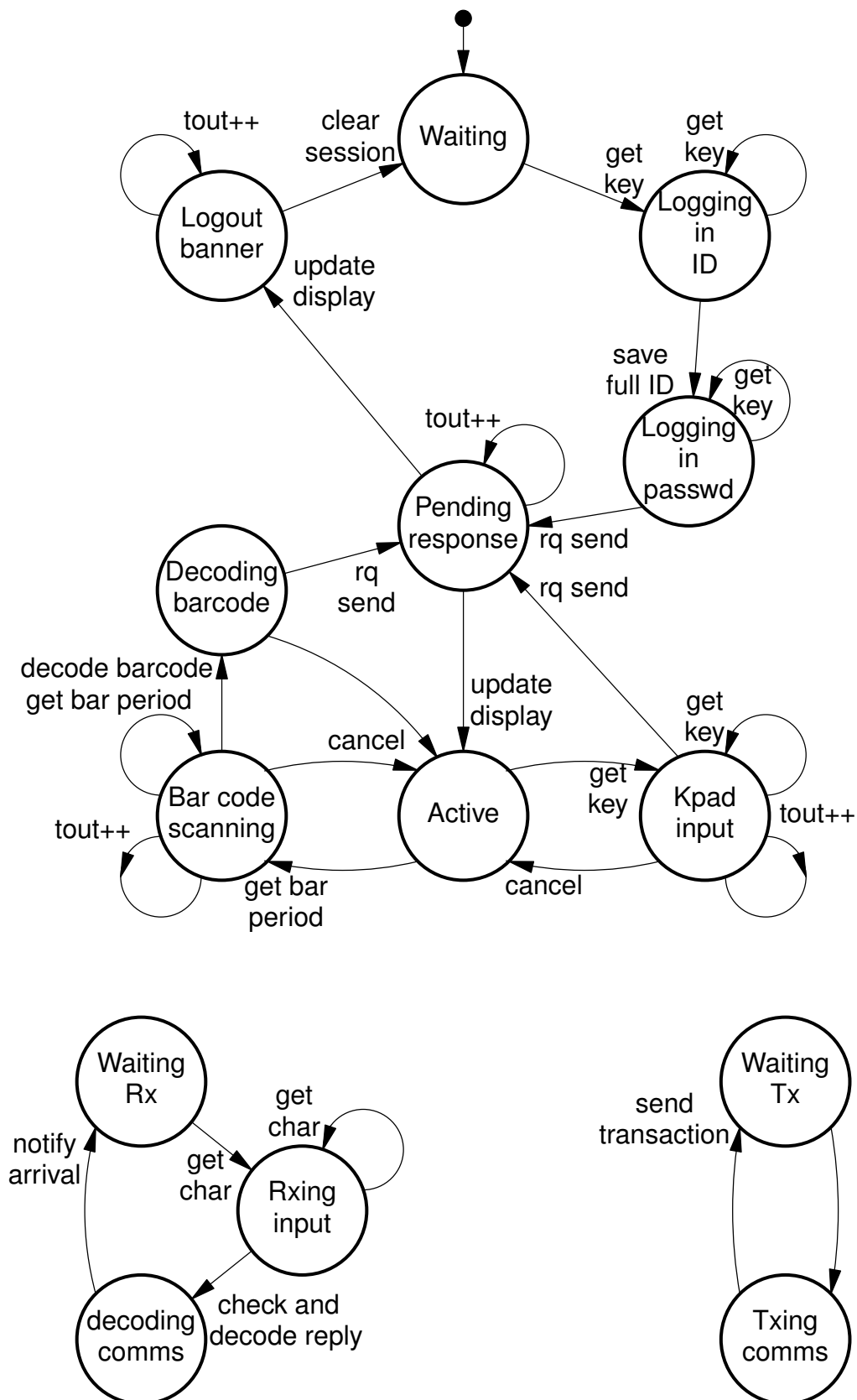
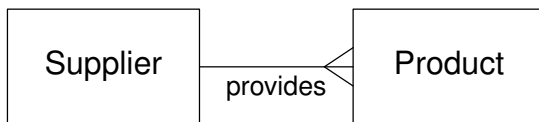


Fig. 11.9d Three concurrent FSMs for a POS terminal, from Fig. 11.6f,

Keypad & Scanner, Serial Rx & Tx



Each supplier may provide several products.

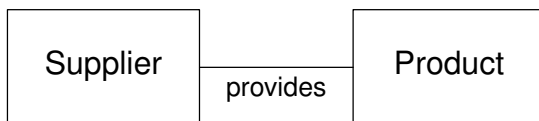
Name

Tel no
Address

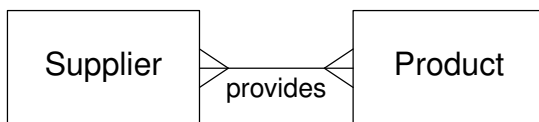
Prod no

Name
Unit cost
Supplier name

Attributes, with the primary key highlighted



Each supplier only provides one unique product rather an unusual situation!



Each product may be provided by several alternative suppliers.

Fig. 11.10a Entity Relationship Diagrams

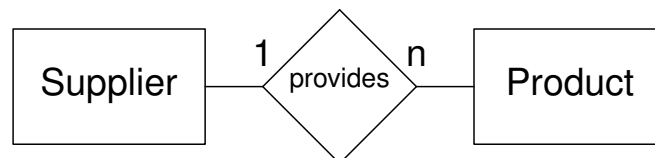


Fig. 11.10b Alternative nomenclature for the ERD

Supplier Name	Tel no	Address
Smiths	123 4567	The Brewery, AX1 2XA
CojaCola	321 7654	Drinks Factory, MS1 2SM
Mcvista	789 0123	Biscuit Corner, RN1 2NR
Tartan	678 9876	The Bakery, SA1 2AS

Product Number	Name	Price	Supplier
01296891	Keg Ale	540	Smiths
01298650	Brown Stuff	125	CojaCola
01273214	Crackers	75	Mcvista
01274521	Oats	45	Tartan
01293245	Lager	225	Smiths
01291552	Cider	185	Smiths
01273221	Digestives	85	Mcvista

Product Number	Name	Price	Supplier
01296891	Keg Ale	540	Smiths
01298650	Brown Stuff	125	CojaCola
01273214	Crackers	75	Mcvista
01274521	Oats	45	Tartan

Product Number	Name	Price	Supplier
01296891	Keg Ale	540	Smiths
01298650	Brown Stuff	125	CojaCola
01273214	Crackers	75	Mcvista
01274521	Oats	45	Tartan
01274521	Oats	45	Mcvista
01293245	Lager	225	Smiths
01291552	Cider	185	Smiths
01273221	Digestives	85	Mcvista
01296891	Keg Ale	540	CojaCola
01273214	Crackers	75	Tartan
01274521	Lager	225	CojaCola

Fig. 11.10c Supplier and product relation tables representing the three alternative relationships from Fig. 11.10a

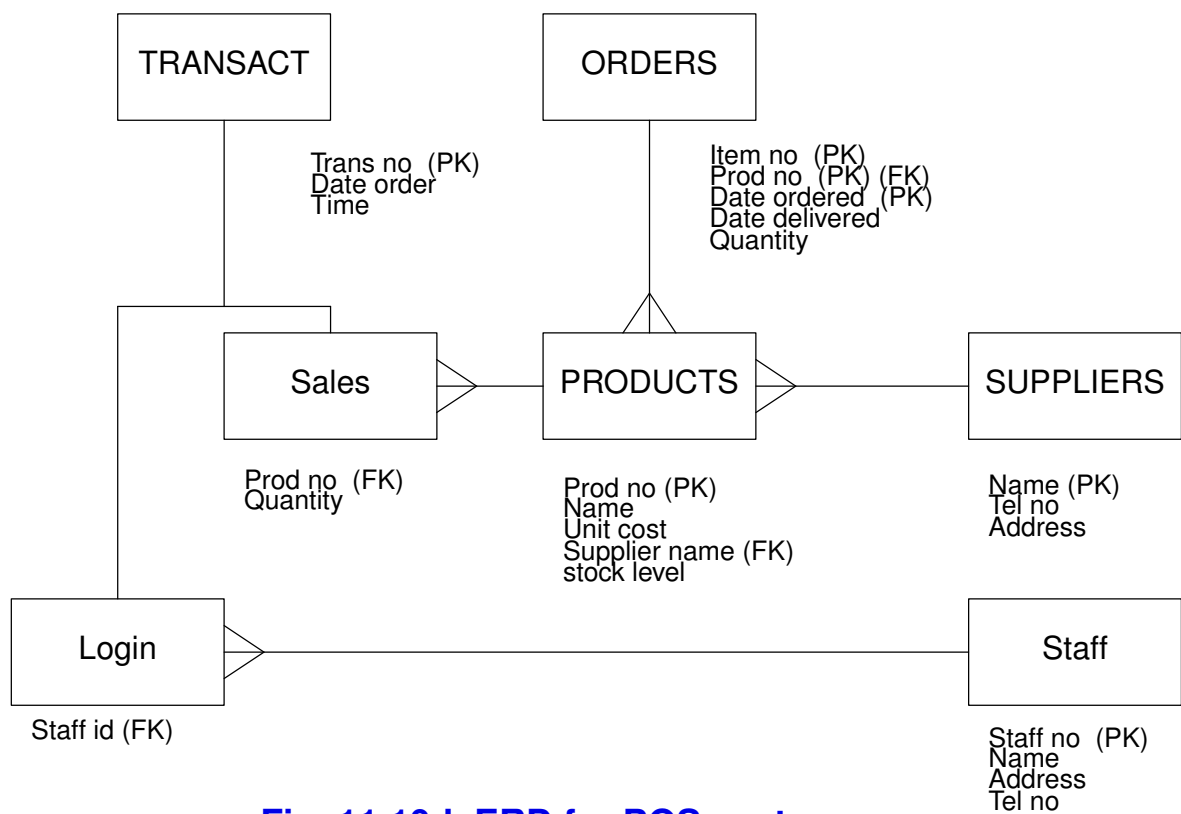


Fig. 11.10d ERD for POS system

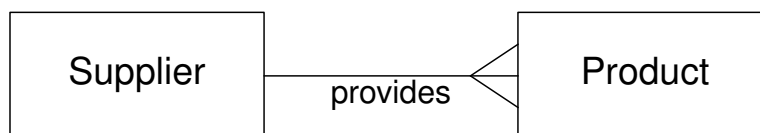
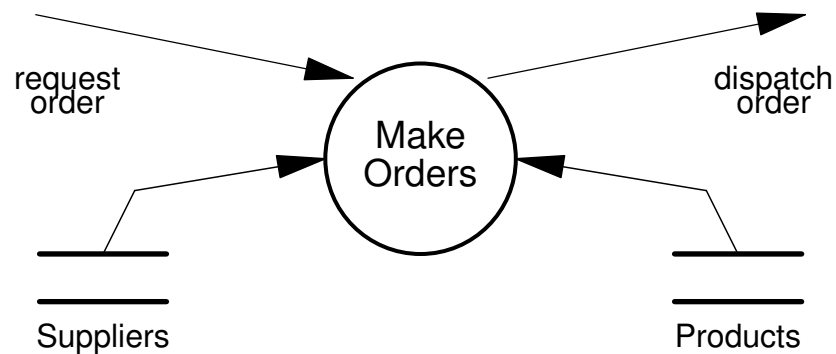


Fig. 11.11 Transforming ERD into DFD

- 1NF First Normal Form, item instances listed in simple attribute relation table may contain redundant data.
- 2NF Second Normal Form, same as 1NF but with non-prime attributes fully dependent on whole prime key attribute(s).
- 3NF Third Normal Form, same as 2NF but with no functional dependencies between non-prime attributes.
- BCNF Boyce-Codd Normal Form, same as 3NF, but with multiple prime key relation tables, the keys are inter-dependent.
- 4NF Fourth Normal Form, same as BCNF but no multi-valued dependencies
- 5NF Fifth Normal Form, same as 4NF but with no non-prime dependencies

12. Designing for Multitasking

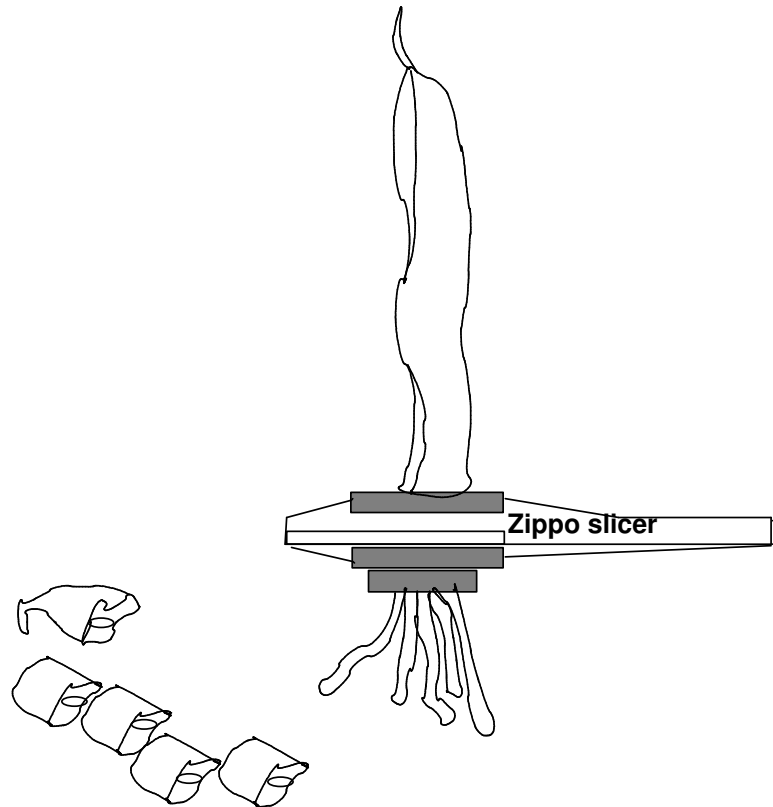


Fig. 12.2 Chopping or slicing the beans?

- platform constraints
- concurrent functionality
 sequential & parallel
- simultaneous I/O activity
- response priorities
- heavy algorithms
- periodic activities
- overlapping for efficiency
- data transfer streams
- possibility of blocking
- exception trapping
- testing purposes

Fig. 12.3 Guidelines for partitioning into tasks

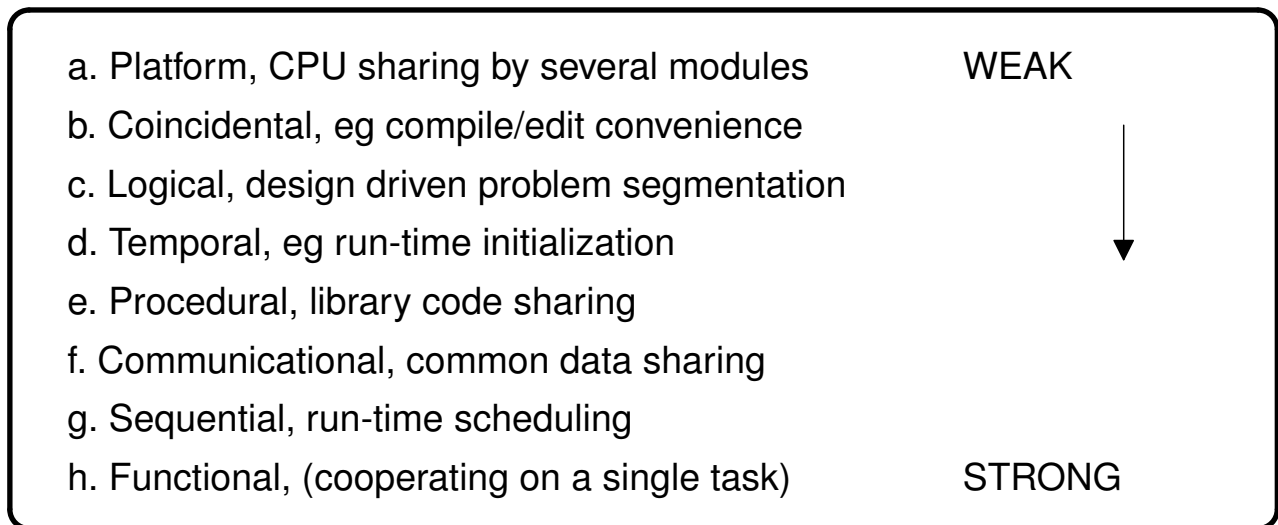


Fig. 12.4a Within Module Cohesiveness
(intra-module cohesiveness)

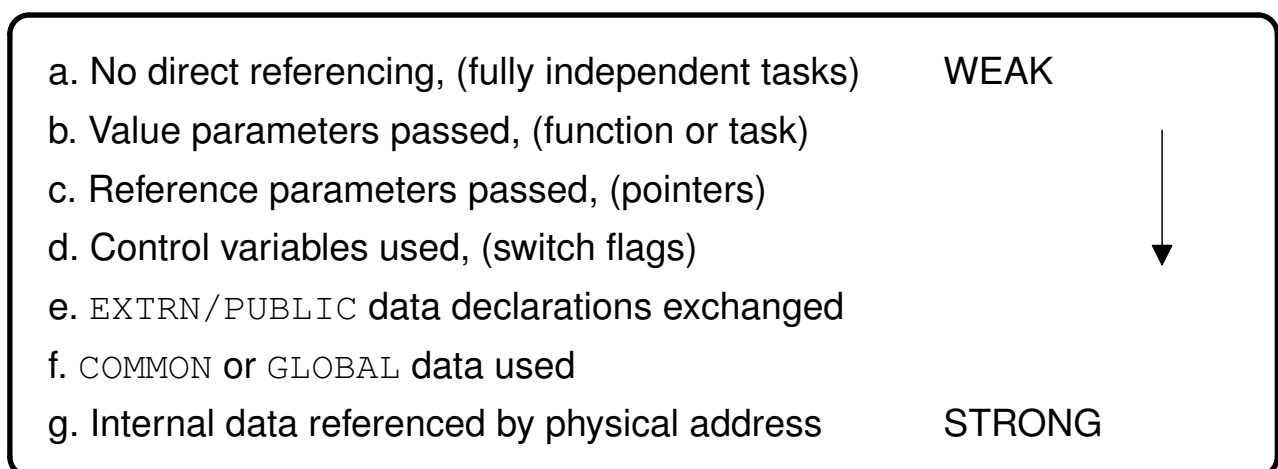


Fig. 12.4b Between module coupling
(inter-module coupling)

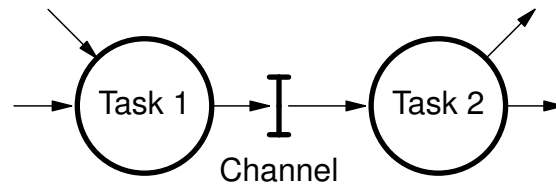


Fig. 12.5a FIFO communication channel between tasks

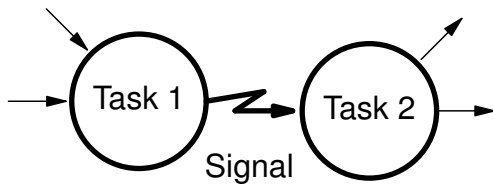


Fig. 12.5b Signaling between tasks

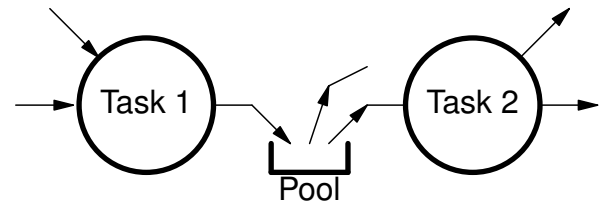


Fig. 12.5c Pool storage buffer

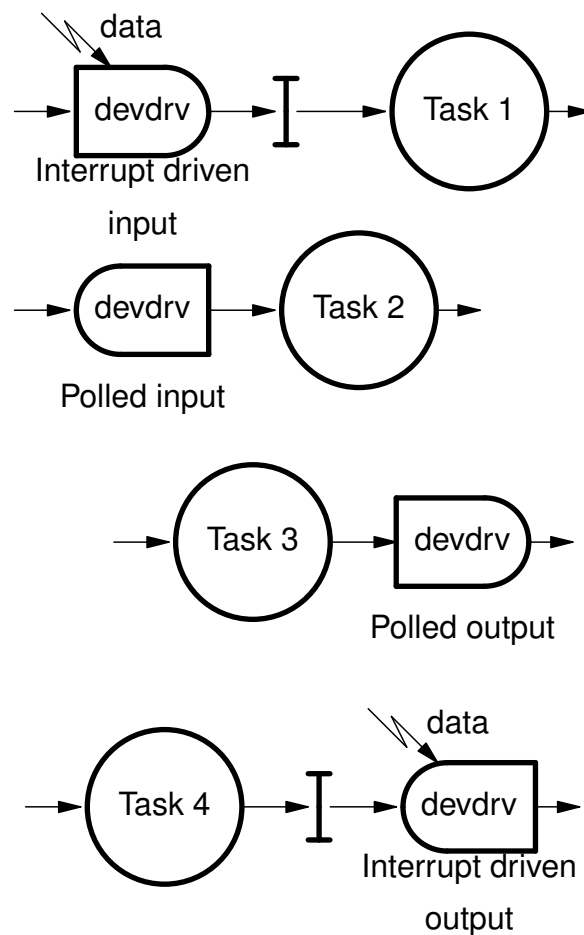
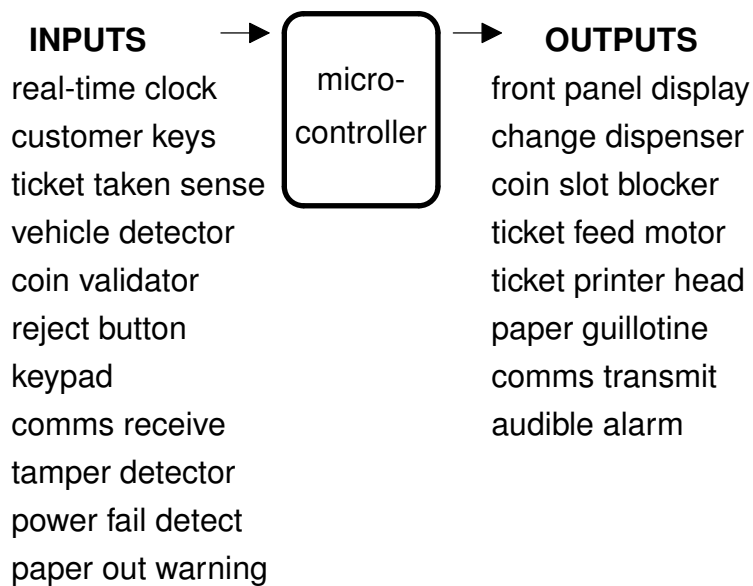


Fig. 12.6d Device drivers



Input	Type	Rate Hz
real-time clock	bit	0.2
customer keys	bit	0.01
ticket taken sense	bit	0.01
vehicle detector	byte	0.3
coin validator	byte	1
reject button	bit	1
keypad	byte	0.3
comms receive	byte	10K
tamper detector	bit	1
power fail detect	bit	100
paper out warning	bit	1
Output		
front panel display	bit	0.01
change dispenser	bit	10
coin slot blocker	bit	0.2
ticket feed motor	bit	0.2
ticket printer head	byte	200
paper guillotine	bit	1
comms transmit	byte	10K
audible alarm	bit	1

Fig. 12.7a Context Diagram for a Pay & Display ticket vending machine

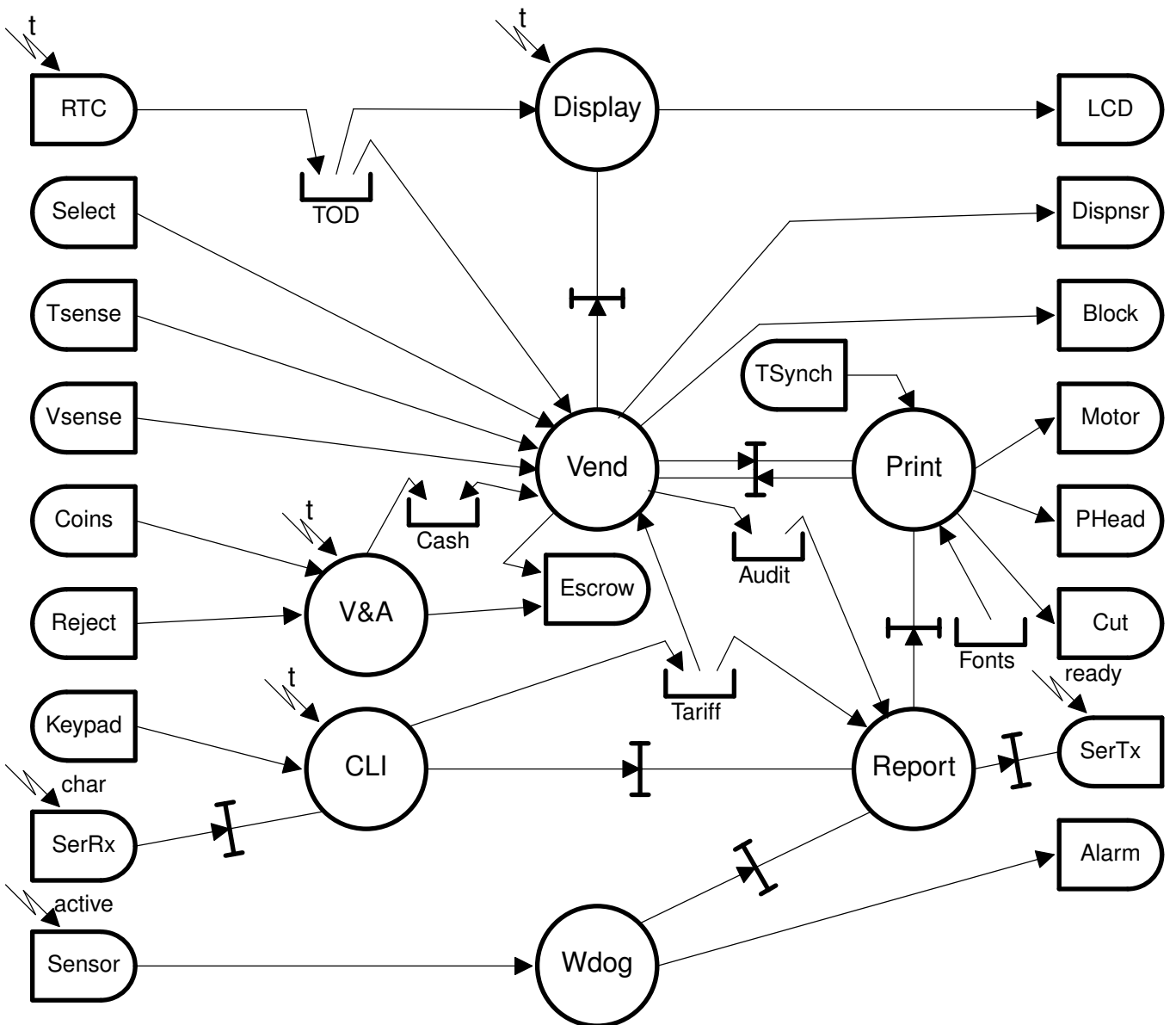


Fig. 12.7b Task diagram for a Pay and Display ticket vending machine

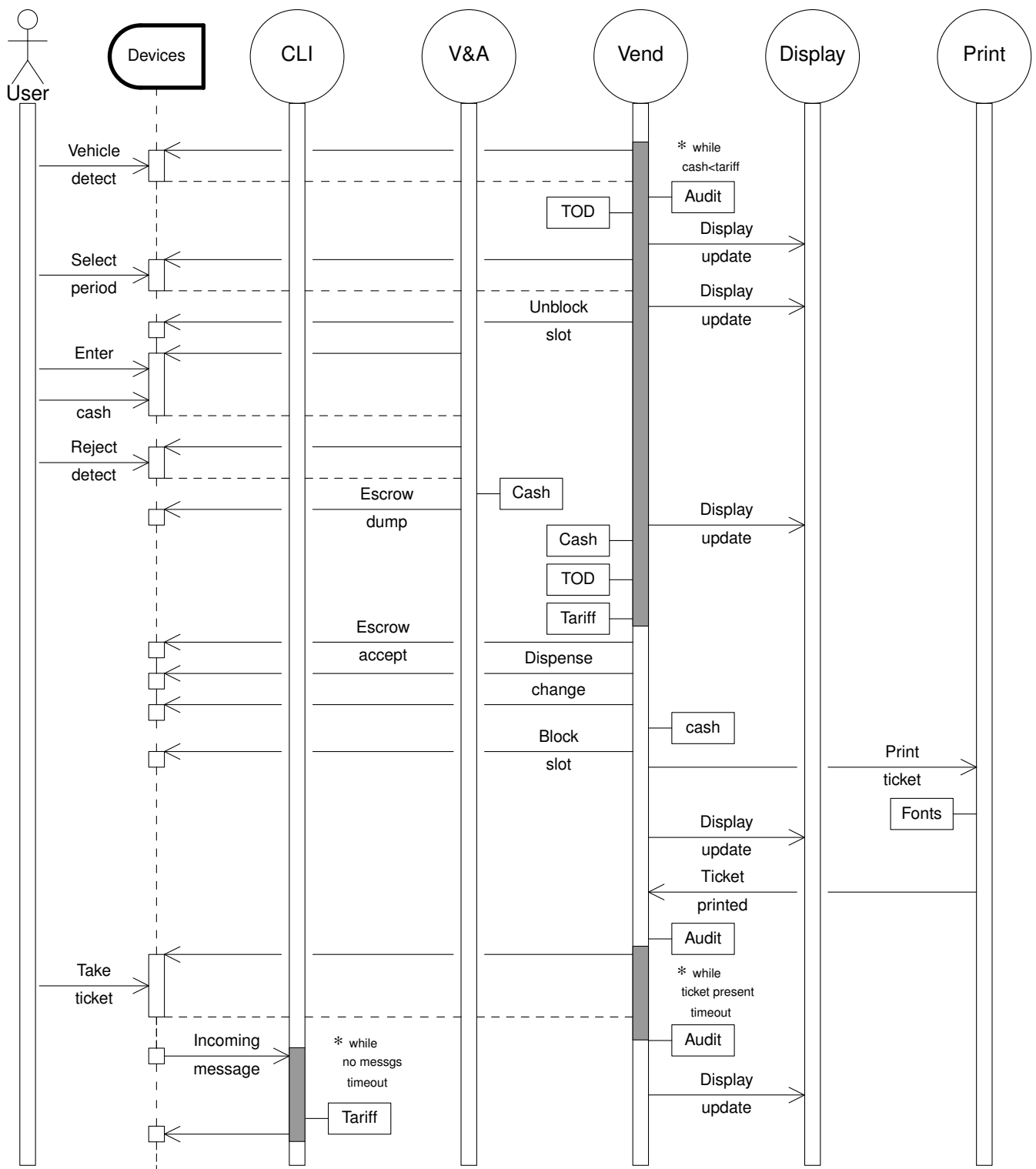


Fig. 12.7c Sequence diagram for a Pay & Display ticket machine

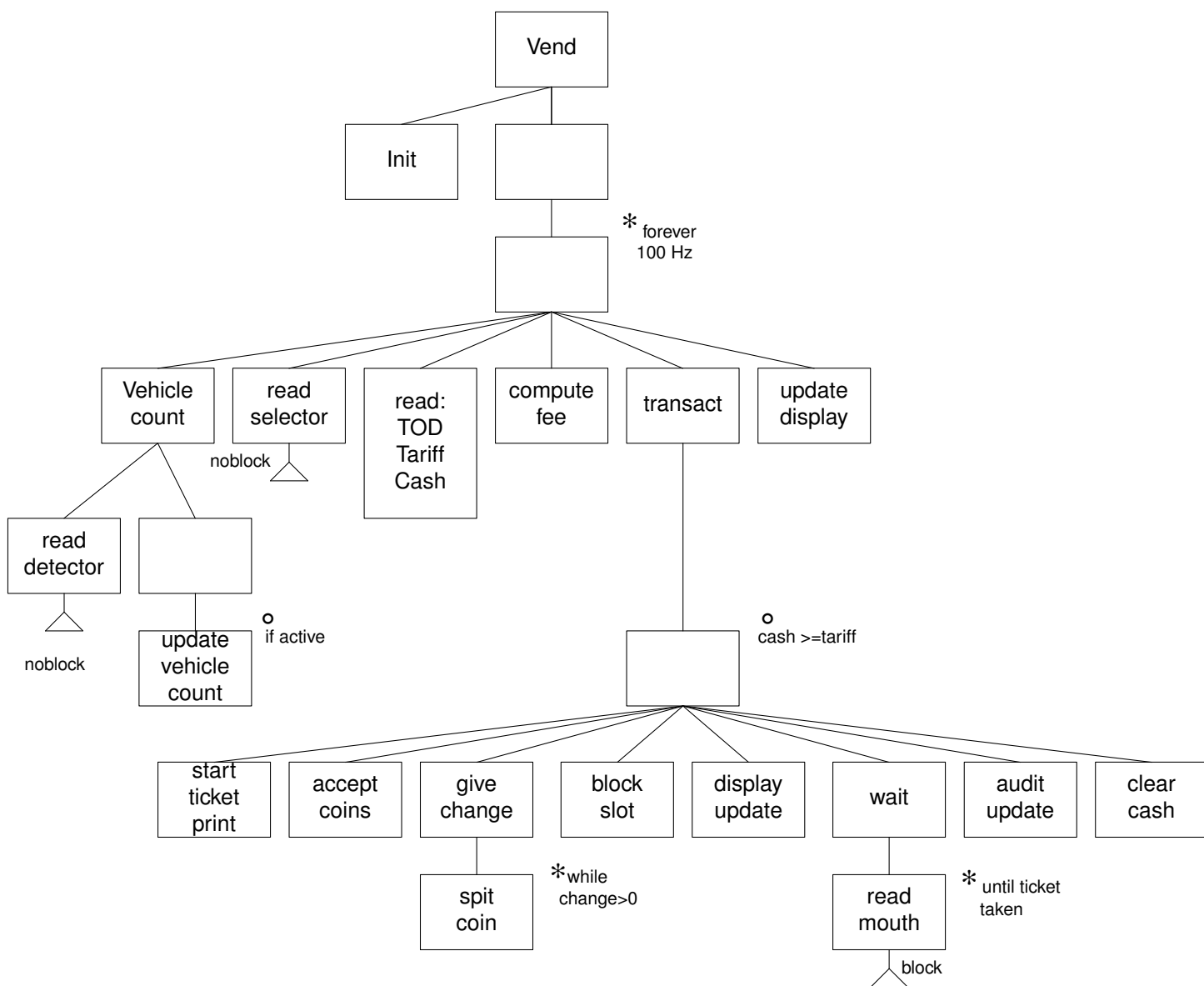


Fig. 12.7d Structure chart for the PnD Vend task

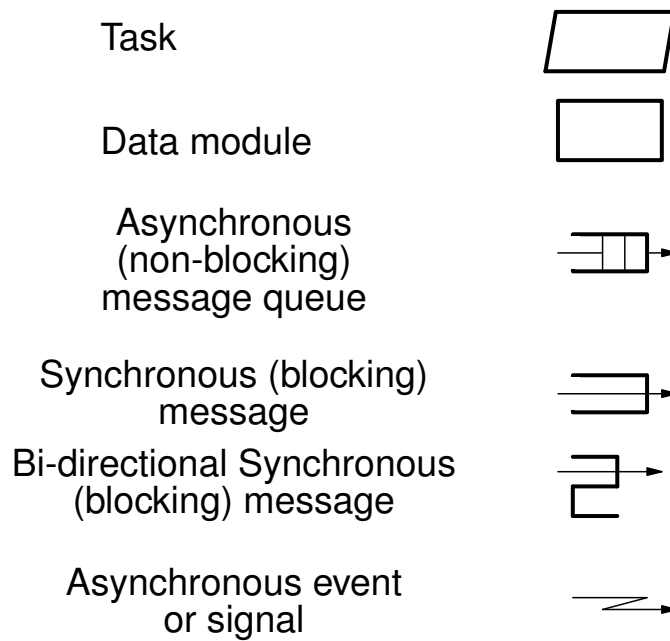


Fig. 12.8 Elements for DARTS task diagrams

- Draw a System Context diagram and hierarchically decompose into subsystems of DFDs for allocation to platforms. Define the behavioural characteristics of each component. Similar to SA/SD methods.
- Identify and form the principal tasks, based on standard criteria as listed in Section
- Identify the message channels and where the need for synchronization primitives may occur between the schedulable tasks
- Establish the significant, enduring application data which has to be stored in data-hiding modules, establish their initialization and access functions.
- Task and data module inter-relationships are defined.
- Fully define all component interfaces, including data and control flows.
- Incremental software development within the separate tasks using structure diagrams.

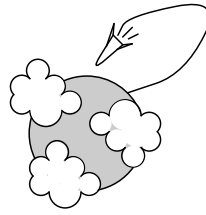


Fig. 12.9 Reentrant code problems

- Algorithm estimation
- Simulation run
- Instruction audit
- Software probes
- Physical measurement

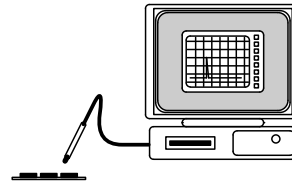
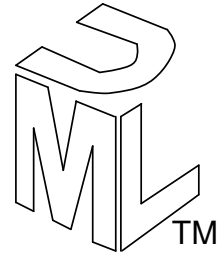


Fig. 12.11 Predicting & Measuring execution times

13. UML for RTS



- Use-case Diagram
- Object Collaboration Diagram
- Class Diagram
- Sequence Diagram
- Statechart (FSM)
- Implementation Diagram
- Activity Diagram

Fig. 13.2 UML Diagrams

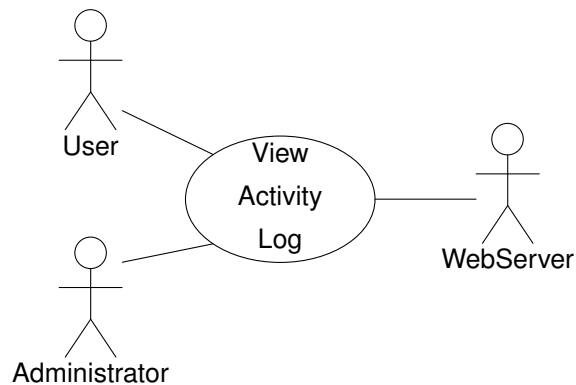


Fig. 13.3a Example use-case diagram

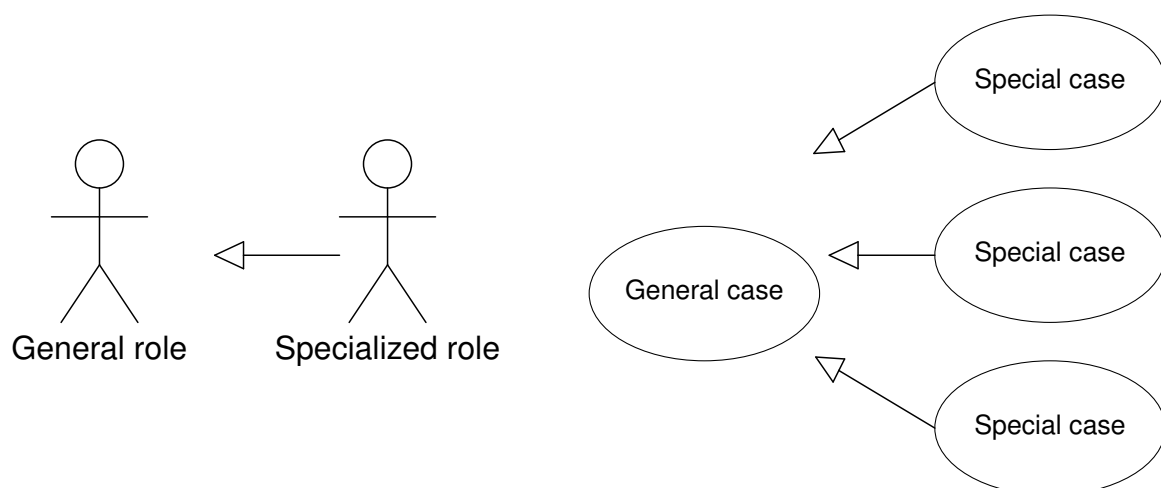


Fig. 13.3b Use-case Generalizations

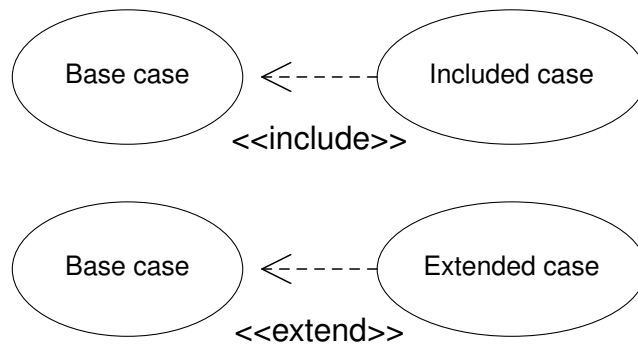


Fig. 13.3c Use-case extension and inclusion

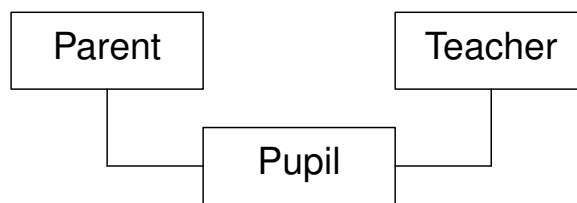
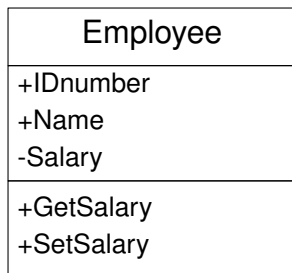
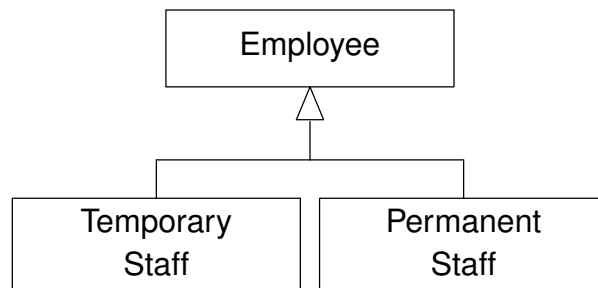


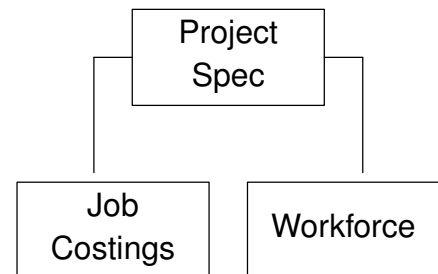
Fig. 13.5 Object Collaboration Diagram



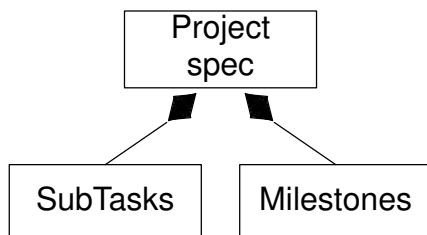
**Class
Component**



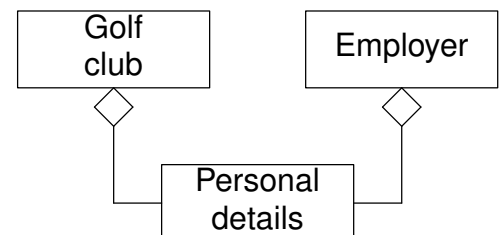
**Derived Class
Generalisation**



Association



Composition



Shared Aggregation

Fig. 13.6a-e

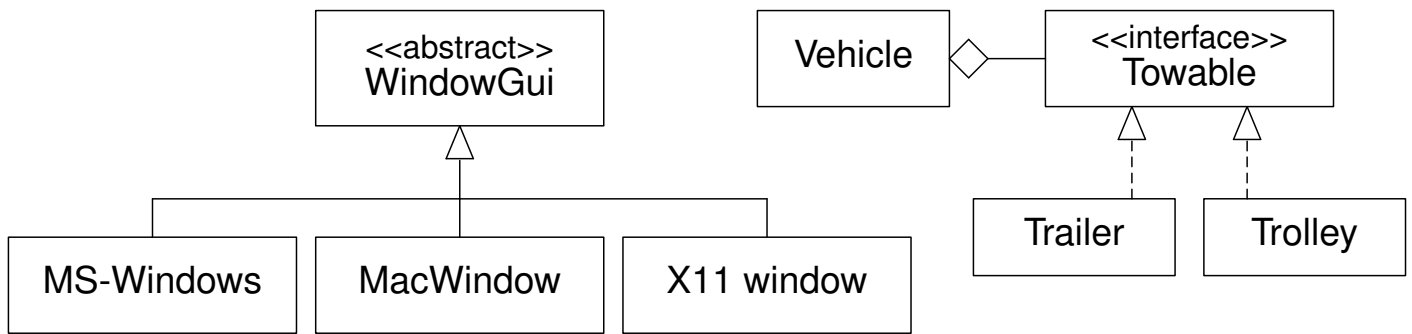


Fig. 13.6f Abstract Base Class

Fig. 13.6g Implementing an interface

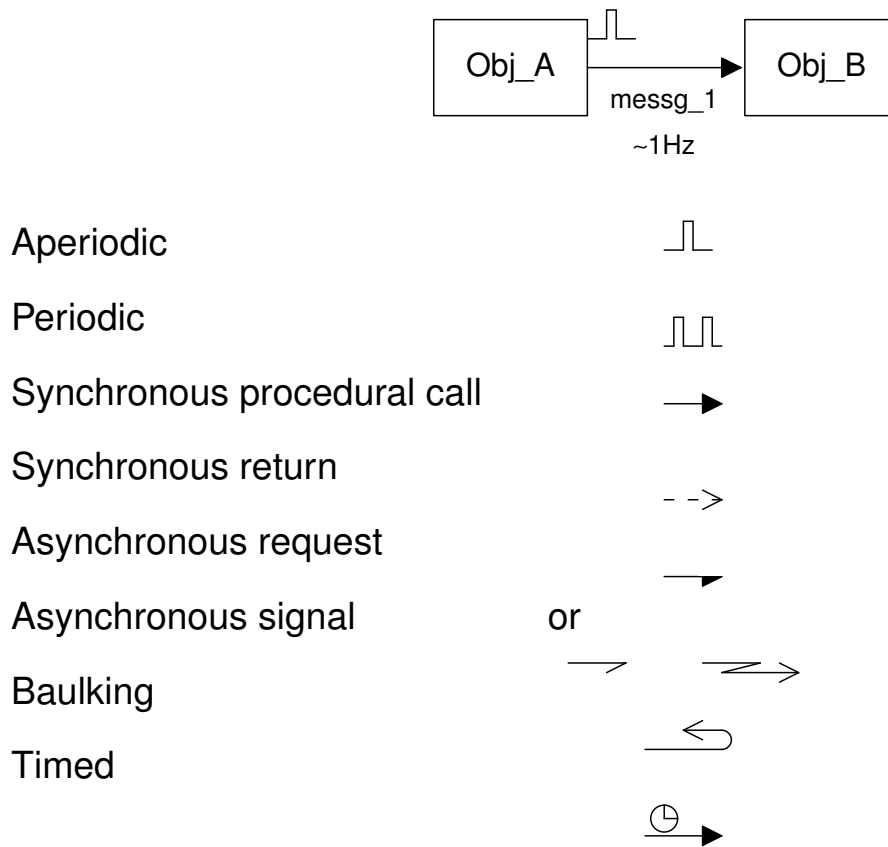


Fig. 13.7 UML Message Types

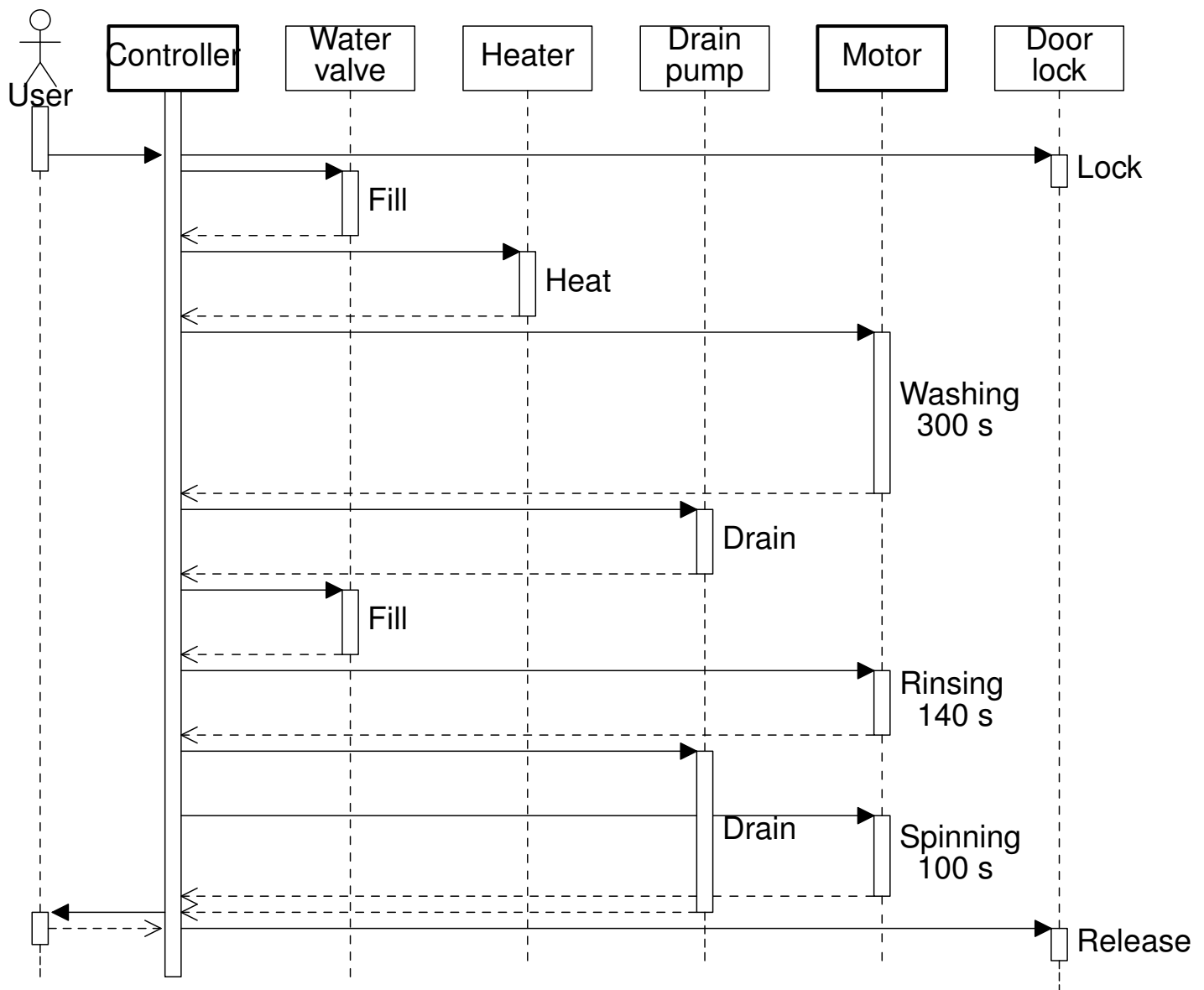


Fig. 13.8a Object Sequence Diagram for a washing machine

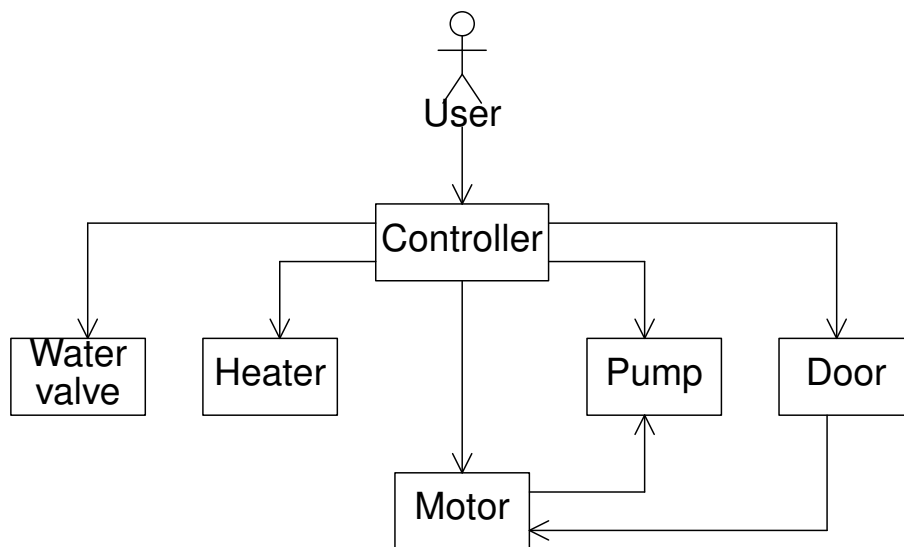


Fig. 13.8b Object collaboration diagram for a washing machine

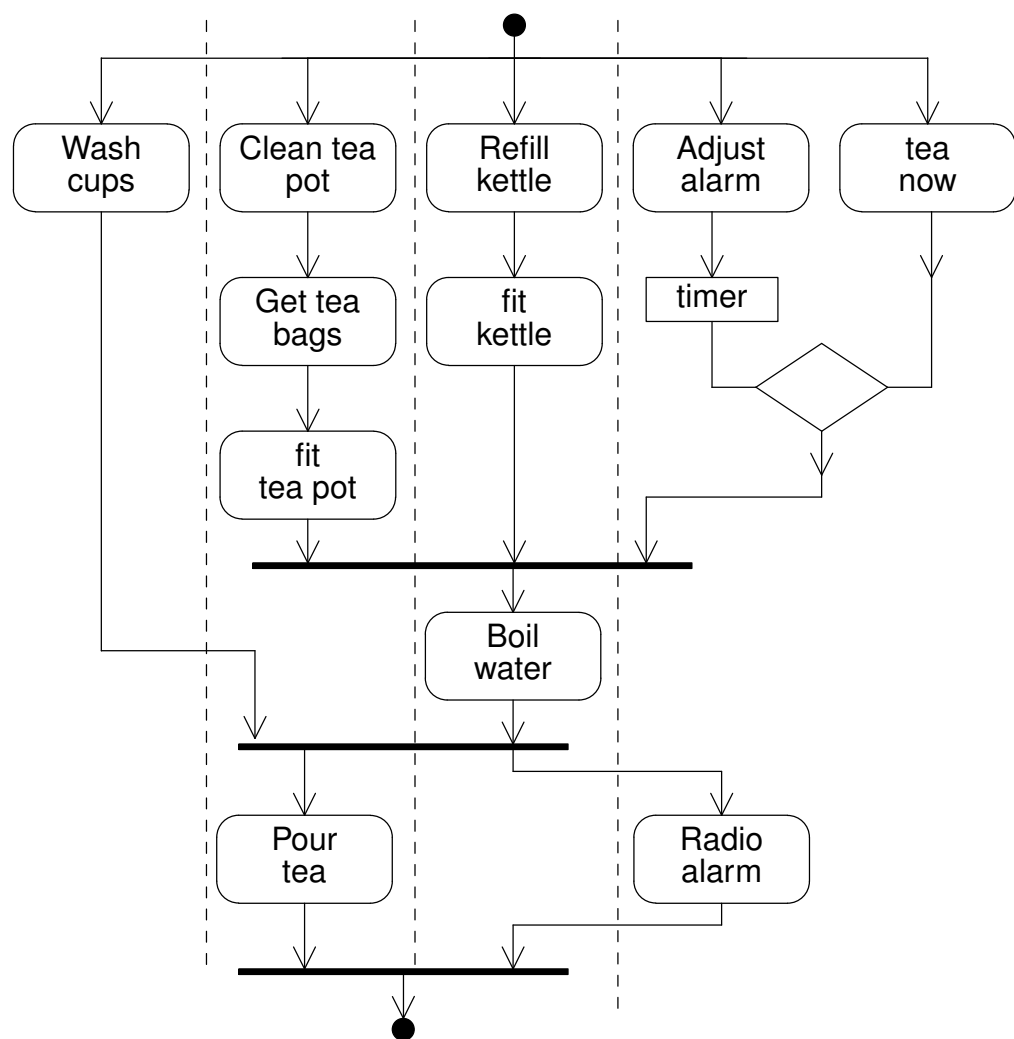


Fig. 13.9 Activity Diagram displaying Swim Lanes

14. Object Oriented Approach for RTS

Activity	Process	Deliverables	
Analysis	Requirements analysis	Use Case models with scenarios	Use case diagrams with descriptive text Message lists
	Systems analysis	High-level initial Architectural model Refined control algorithms	Class diagrams at the subsystem level Component diagrams Activity diagrams
	Object structural analysis	Structural obj model	Class diagrams Object diagrams
	Object behavioural analysis	Behavioural obj model	Object sequence diagrams Statecharts
Design	Architectural design	Concurrency model Deployment model Component model	Identify active objects Deployment diagrams O/S tasking model
	Mechanistic design	Collaboration model Message seq diagrams	Class diagrams Patterns State execution model
	Detailed design	Class details	Methods, attributes, user-defined types, package members
Transln		Executable application	Fully executable code generated from structural and behavioural models,
Testing	Unit testing Systems integration & validation	Code corrections Design defects list	Design-level debugging Test reports

Fig. 14.2 Object-oriented software lifecycle

Identify ->	
key nouns	This suits when a written specification is available
causal sources	Sources of events, actions, messages
services	Targets for events or messages, providing actions for others
real world items	The system will be a real world model
devices	Terminal equipment such as sensors and actuators
concepts	Such as bank account, subscription, signal spectrum
transactions	Objects are bound together by transactions
data	Persistent data, becoming an object's attribute

Fig. 14.5 Methods to help identify useful objects

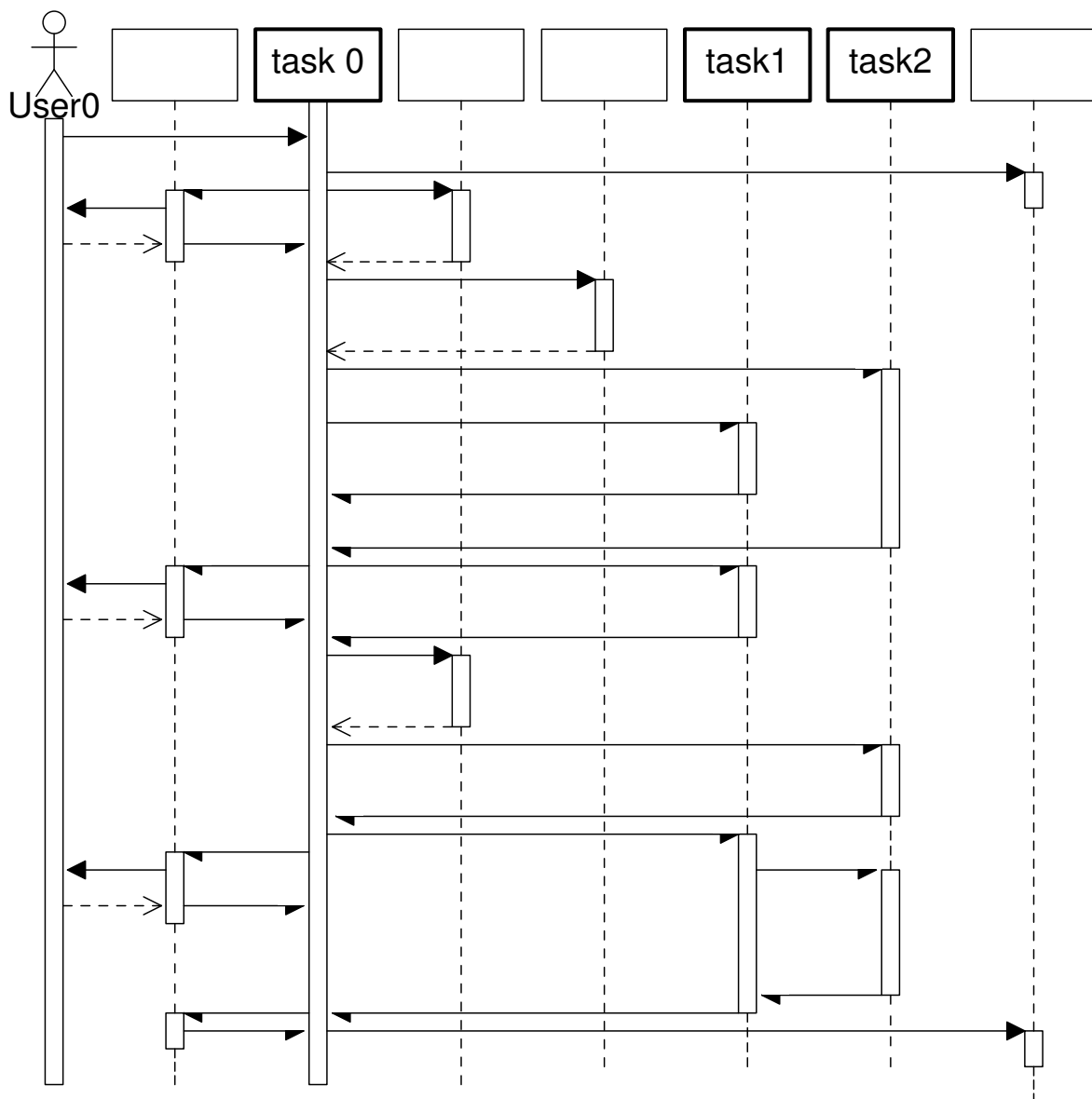


Fig. 14.7 Sequence diagram with asynchronous messages between concurrent tasks

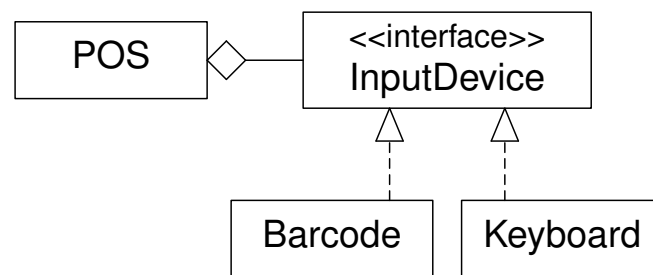


Fig. 14.9a Implementing an interface

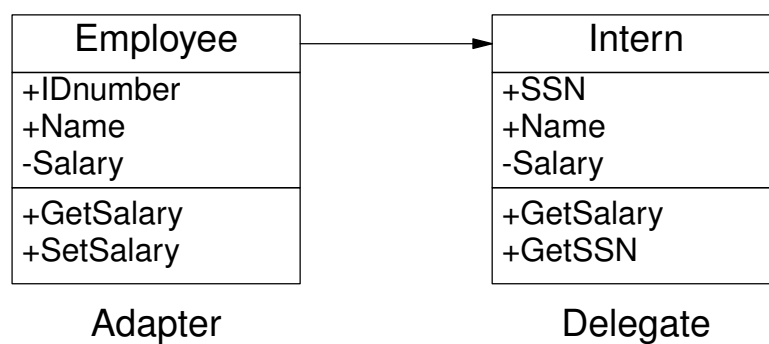


Fig. 14.9b Facade pattern

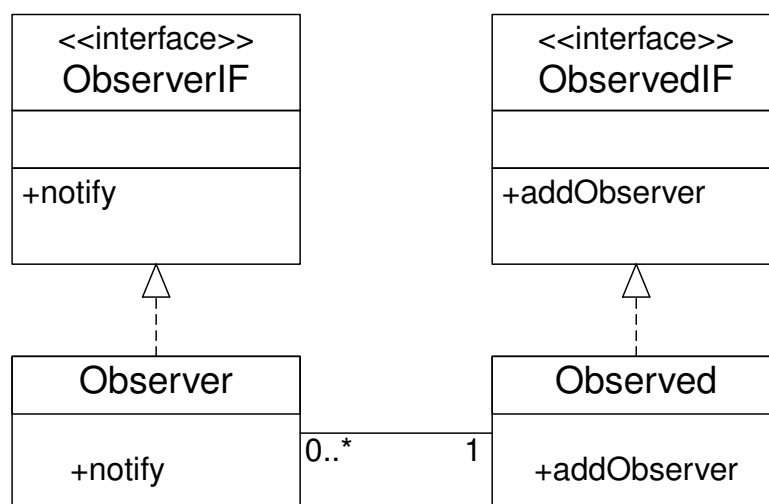


Fig. 14.9c Implementing an observer pattern

Factory patterns	
Builder	simple factory class generates instances of <i>concrete</i> classes
Abstract factory	multiple factory classes for a common set of objects
Flyweight	constructing shared objects
Singleton	limits object instantiation by hiding the constructor
Factory method	putting the factory inside another class
Prototype	cloning new objects
Algorithmic patterns	
Strategy	using an object to contain an algorithm
Template	subdividing an algorithm into fixed and variable parts
Memento	saving and restoring the internal state of an object
Delegation patterns	
Interface	Abstract interface to all methods
Adaptor	delegating functionality to another object
Bridge	treating a class as a pointer to another
Decorator	using delegation to include new behaviour
Facade	creating subsystems of objects, encapsulated together
Proxy	placeholders for remote objects
Control patterns	
Composite	using objects to achieve groupings
Interpreter	interpreting a command language
Command	reorganizing commands
Iterator	stepping through a composite structure
Visitor	providing indirect methods in a class hierarchy
Mediator	controlling the interaction of other objects
Observer	publish-subscribe messaging framework

Fig. 14.9d FADC, categorization of design patterns,
based on Reiss [1999]

15. System Integrity

"Quality is remembered long after
the price is forgotten"

"The memory of bad quality lasts longer
than the shock of high prices"

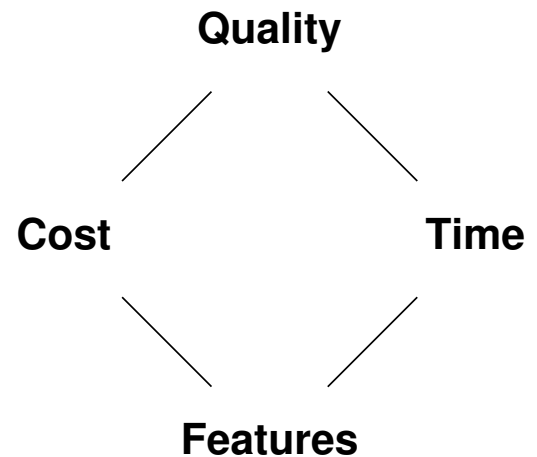


Fig. 15.2a Conflicting commercial pressures

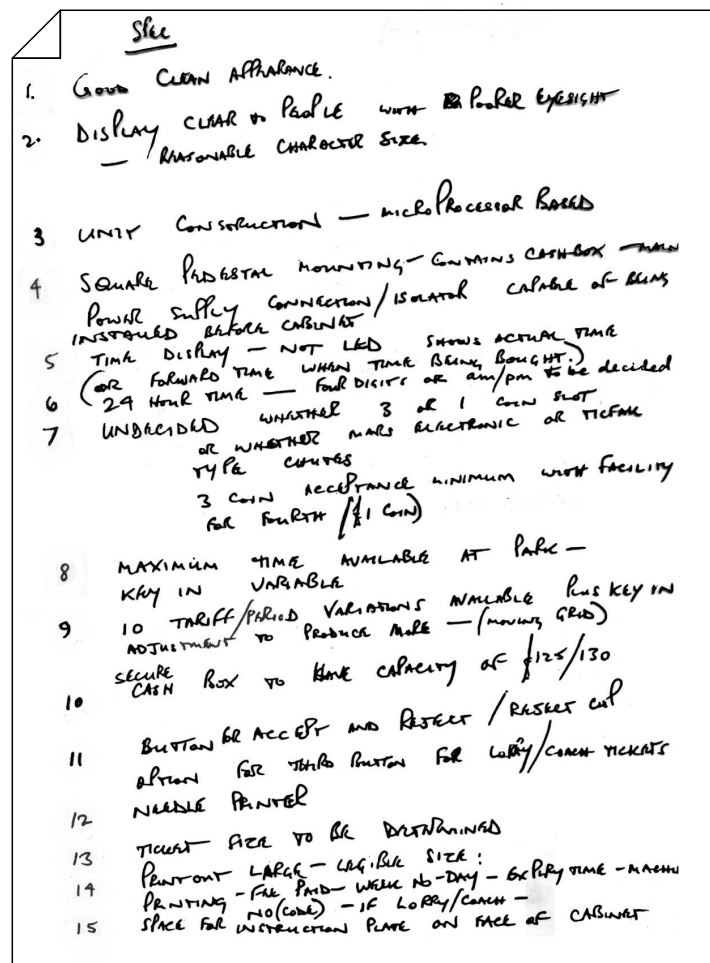


Fig. 15.2b A real requirements document

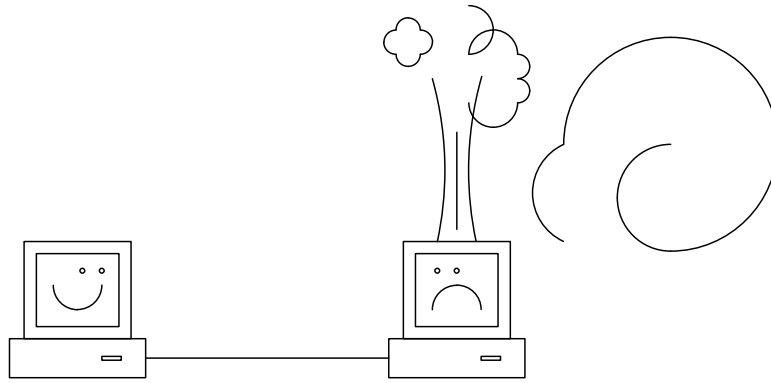


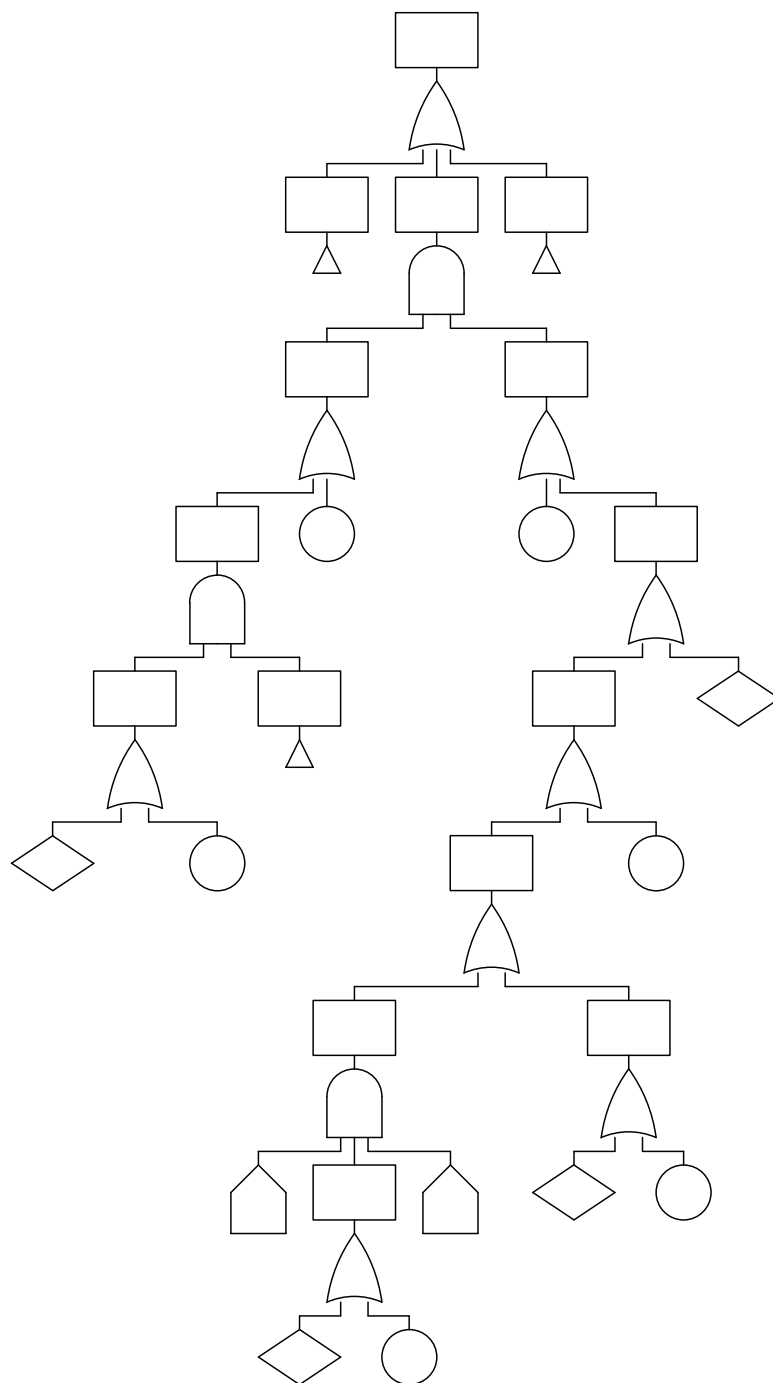
Fig. 15.3a A small server problem

	Availability per year	Annual minutes downtime	
3x9s	99.9%	526.6	
4x9s	99.99%	52.56	1 hour
4.5x9s	99.995%	26.28	1/2 hour
5x9s	99.999%	5.256	5 minutes
6x9s	99.9999%	0.5256	30 secs
	100%	0	

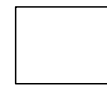
Fig. 15.3b Availability figures

1. Positive & negative specifications
 - what the system should do & shouldn't do
2. Determine major failure modes
3. Identify mitigation strategies
4. Divide into subsystems to limit error propagation
5. Establish redundant capabilities
6. Plan error management strategy
7. Major system design decisions
8. Interface planning

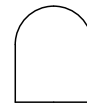
Fig. 15.3c Designing for Fault Tolerance



Basic failure event



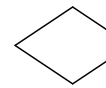
Intermediate error event



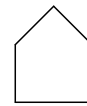
o/p failure IF all i/p failures



o/p failure IF some i/p failure



undeveloped event

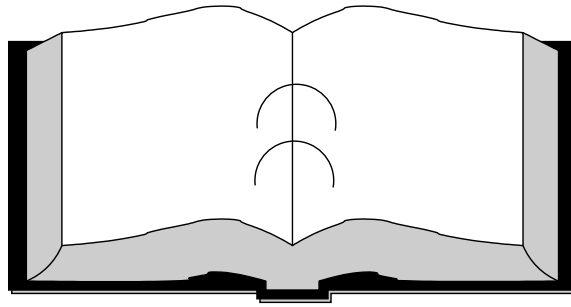


external planned event



connector to other page

Fig. 15.3d Fault tree analysis



- under-resourced activity carried out by inexperienced staff
- complications due to attempting to reuse previous documents
- customers unprepared with only vague ideas
- irreconcilable viewpoints from different departments
- 'hi-tech' enthusiasm from senior managers: feature creep
- design decisions put in too soon
- ambiguous customer descriptions
- customer continually changing ideas
- no commitment to agreeing test data
- hidden agendas or constraints
- clients' requirements too ambitious

Fig. 15.4 Common problems for the Requirements team

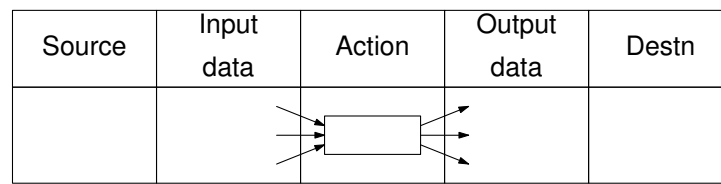


Fig. 15.5a CORE viewpoint diagram

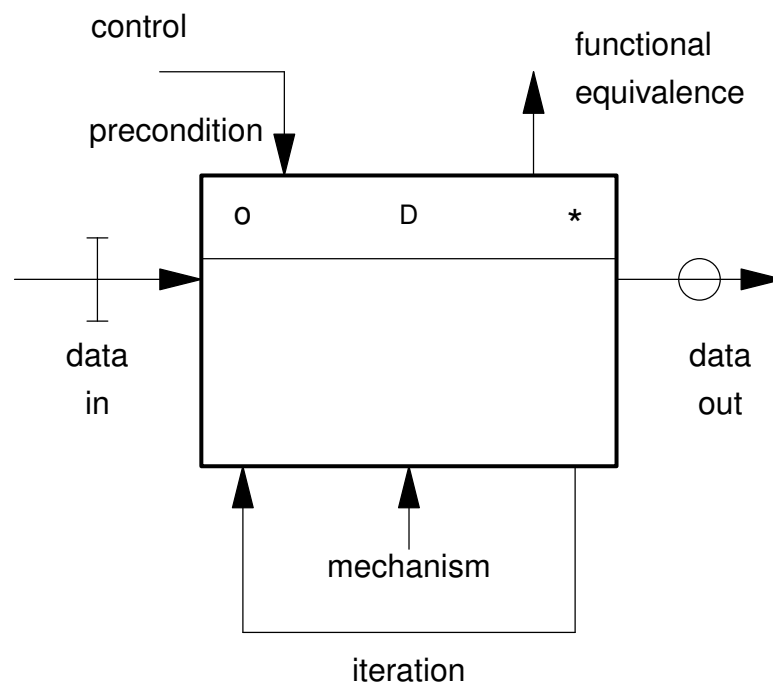


Fig. 15.5b CORE activity diagram

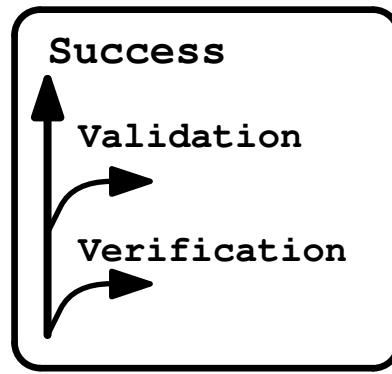


Fig. 15.6 Verification & Validation

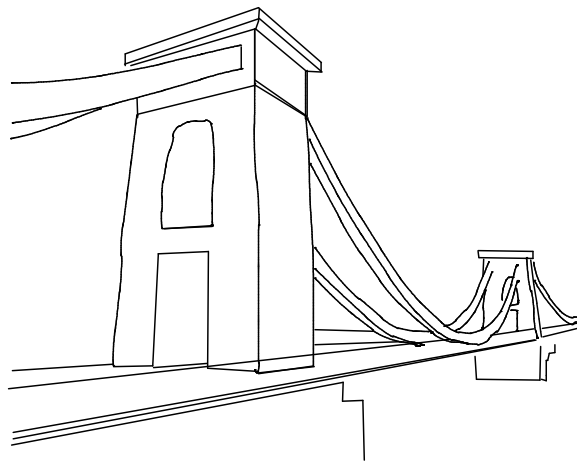


Fig 15.7 Clifton suspension bridge, a good, enduring design by Isambard Brunel (1864)

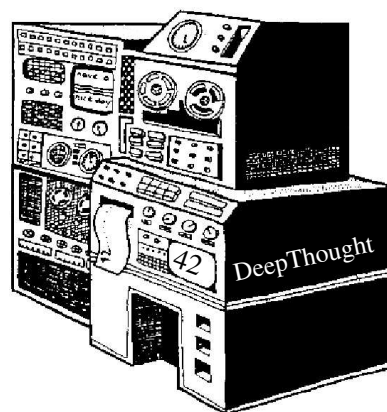


Fig. 15.8 Life, the Universe and Everything? [Adams, 79]

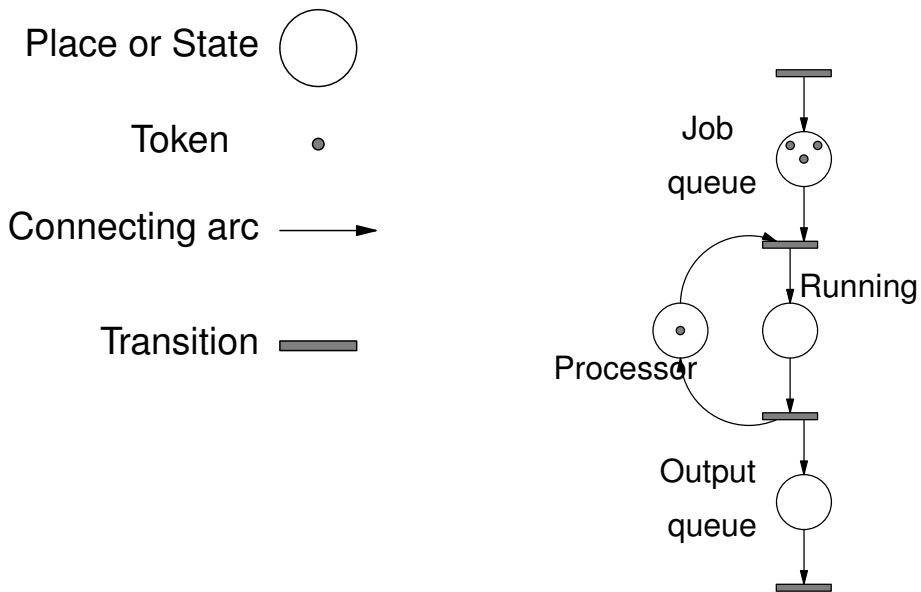


Fig. 15.9a Petri Net diagram for a process queue

Places - possible states of the system
 Transitions - allowed system events, state -> state
 Inputs - preconditions for a transition
 Outputs - postconditions for a transition
 Tokens - shows current system resource status

Fig. 15.9b Petri Net Graph components

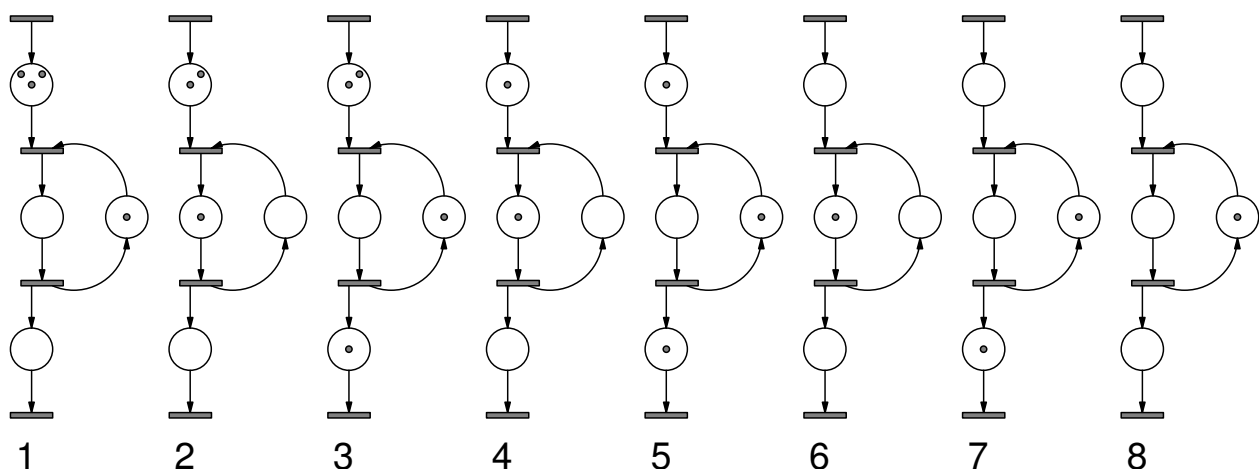


Fig 15.9c A sequence of Petri Net activity with three simultaneous requests for a single exclusive resource

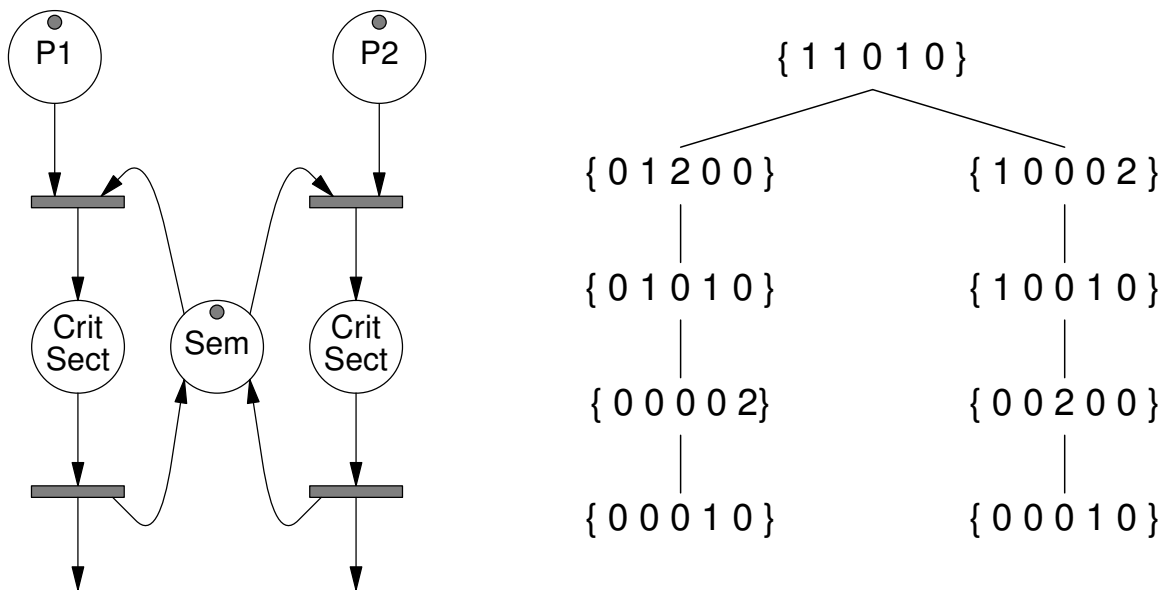


Fig. 15.9d Modelling Semaphore action using Reachability Tree

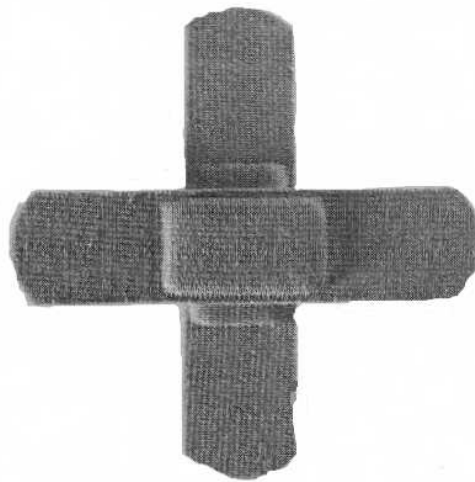


Fig. 15.10 A touch of lint

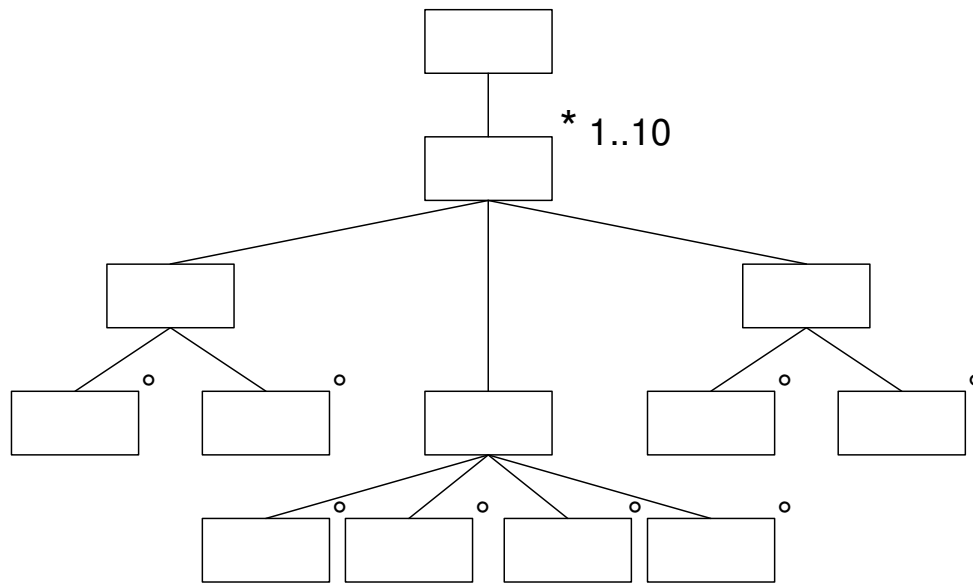


Fig. 15.13 Testing complexity: 10^{12} pathways

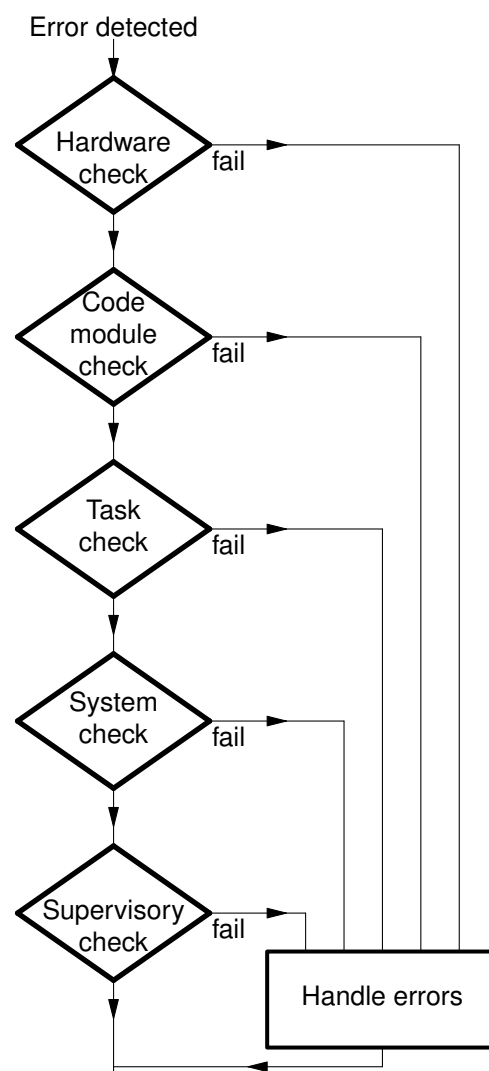


Fig. 15.15 System run-time checks

```
# define assert(expr) ( __ASSERT_VOID_CAST((expr) ? 0 :\
    (__assert_fail(__STRING(expr), __FILE__, __LINE__, __ASSERT_FUNCTION), 0)) )

# ifdef          __USE_GNU
#   define assert_perror(errnum)  ( __ASSERT_VOID_CAST(!(errnum) ? 0 :\
    (__assert_perror_fail((errnum), __FILE__, __LINE__, __ASSERT_FUNCTION), 0)) )
# endif

assert(pdata != NULL);
```

Fig. 15.16 Run-time checking: assert

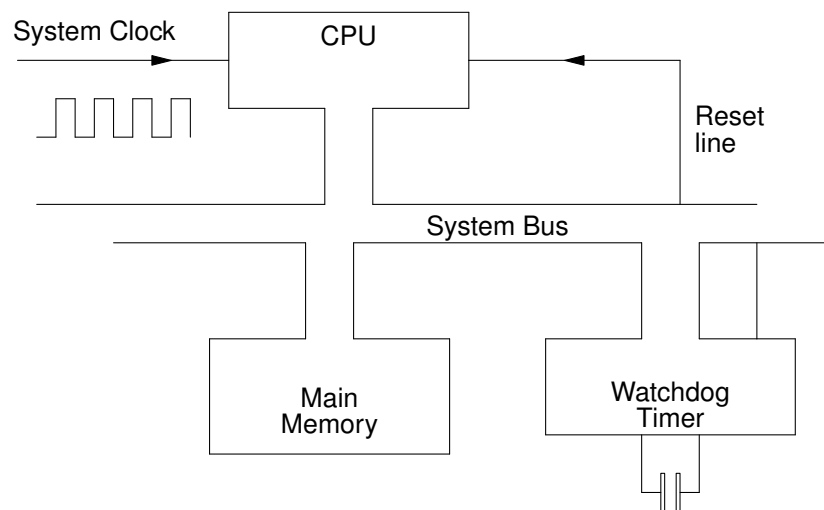


Fig. 15.17 Watchdog Timer

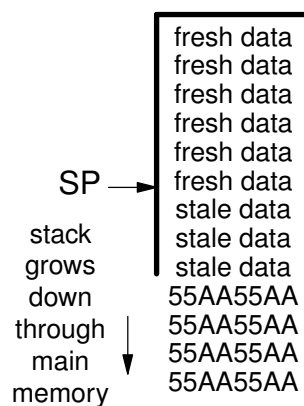


Fig. 15.18 Stack length checking

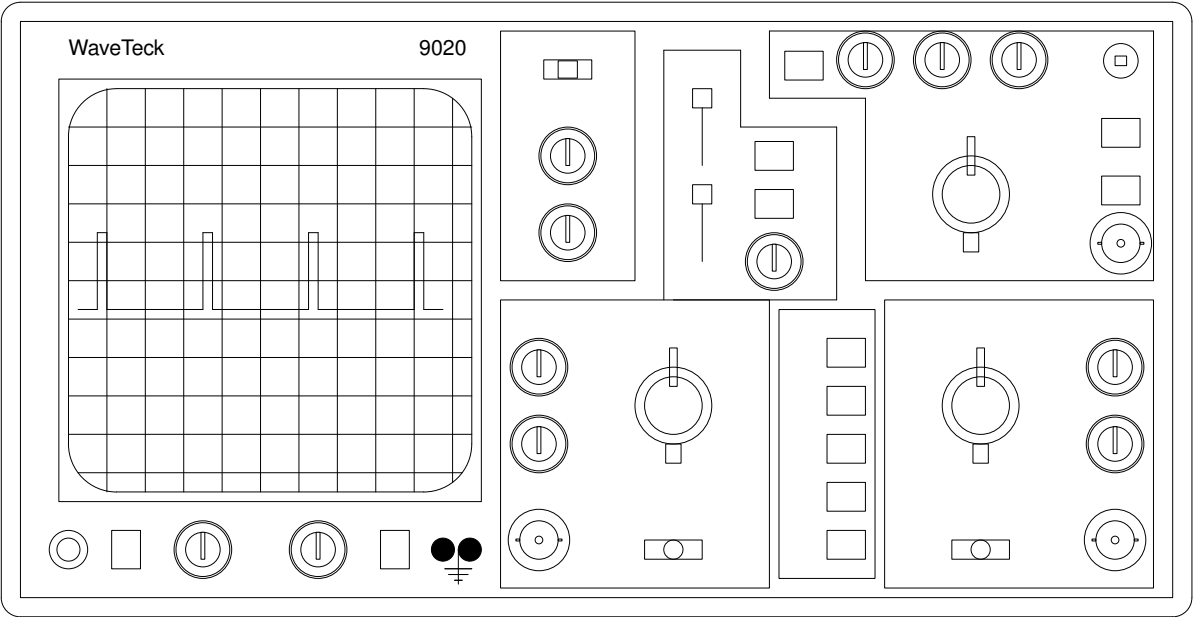
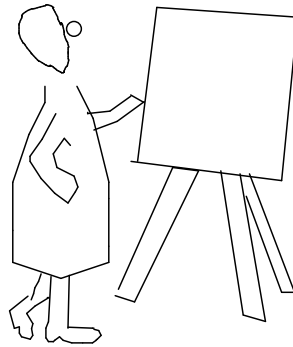


Fig. 15.19 Visual debugging aid

CVS

add	Add a new file/directory to the repository
admin	Administration front end for rcs
annotate	Show last revision where each line was modified
checkout	Checkout sources for editing
commit	Check files into the repository
diff	Show differences between revisions
edit	Get ready to edit a watched file
editors	See who is editing a watched file
export	Export sources from CVS, similar to checkout
history	Show repository access history
import	Import source files into CVS repository
init	Create a CVS repository if it doesn't exist
log	Print out history information for files
login	Prompt for password for authenticating server
logout	Removes entry in .cvspass for remote repository
pserver	Password server mode
rannotate	Show last revision where each line of module was modified
rdiff	Create 'patch' format diffs between releases
release	Indicate that a module is no longer in use
remove	Remove an entry from the repository
rlog	Print out history information for a module
rtag	Add a symbolic tag to a module
server	Server mode
status	Display status information on checked out files
tag	Add a symbolic tag to checked out version of files
unedit	Undo an edit command
update	Bring work tree in sync with repository
version	Show current CVS version(s)
watch	Set watches
watchers	See who is watching a file

```
cvs -d ~/cvsroot/projectA
export CVSROOT=~/cvsroot/projectA.
export CVSAREA=/tmp/projectA.
cvs init
```



- Review the product not the producer -
reduce pressure on those whose work is under scrutiny.
- Stick to the prepared agenda -
don't allow the meeting to be hi-jacked.
- Guillotine debate strictly -
senior staff are busy, overruns are not welcome.
- Summarize problems, don't try to solve them -
discussion on solutions can take place later.
- Make public notes on flip-chart -
decouple review activity from annual appraisals.
- Establish a follow-up procedure -
reduce the administrative variation.
- Be warned and prepare for sessions -
most of us need to read through the material beforehand.
- Train staff for review work -
establish mutual trust.
- Use impersonal tick sheets as far as possible -
routine recording.
- Schedule reviews as part of normal product development -
budget for them.
- Review the reviews -
learn from experience and get better at reviewing.
- All participants sign the formal acceptance -
corporate responsibility.

Fig 15.23 Rules for successful review meetings [Fagan 86]

- Test harnesses produced first
- Pair programming: constant peer review
- Rapid prototyping
- Frequent unit testing
- Code release control (CVS)
- Simple code is best
- No personal code ownership
- Small teams: good communications
- Customer representation

Fig 15.24 Principles of XP

Level		Description
1	Initial	Utter chaos when things go wrong, no agreed s/w standards, poor project management. Product quality unpredictable. Organisation sustained by outstanding staff effort and commitment.
2	Repeatable	Standard processes established and repeatable. Management does carry out retrospective reviews to improve future performance.
3	Defined	The whole organisation has adopted a standardised software development and maintenance process which can be repeated for all s/w projects. Because the procedures are well understood, management has good insight into the day-to-day progress on all projects.
4	Managed	The organisation has adopted quantifiable measures of quality which are applied to both process and product. The process can be adapted in an effective way to suit the needs of each individual project.
5	Optimized	Mature disciplined process which is being continuously refined. Improvement occurs both by incremental advancements in the existing process and by innovations using new technologies and methods.

Fig 15.25a Levels for the Capability Maturity Model

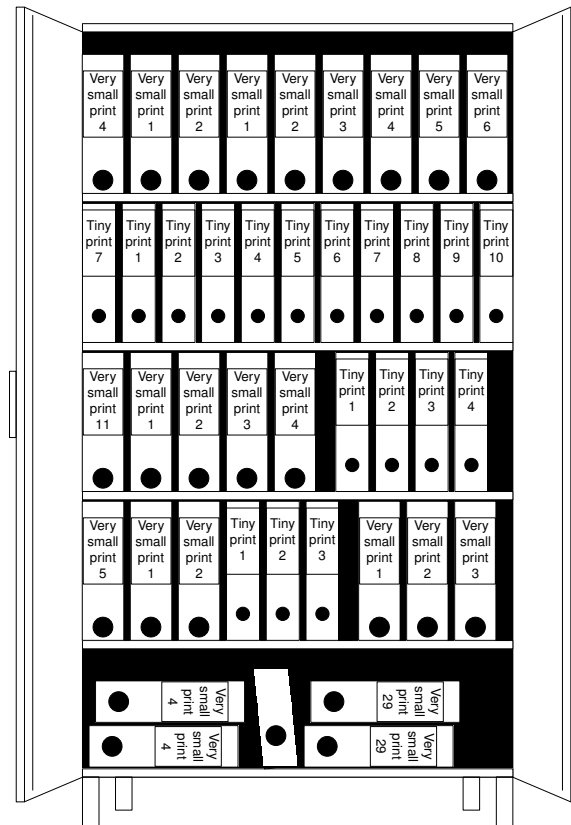


Fig 15.25b MIL-STD-498 documentary trail for a 50 page program

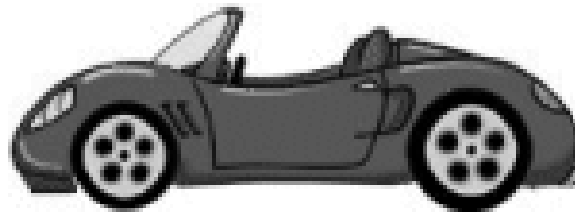


Fig. 15.26 Motor vehicle control software MISRA (Motor Industry Software Reliability Association)

Yesterday,
All those back-ups seemed a waste of pay.
Now my database has gone away.
Oh I believe in yesterday.

Suddenly,
There's not half the files there used to be,
And there's a milestone hanging over me,
The system crashed so suddenly.

I pushed something wrong
What it was I could not say.

Now all my data's gone
and I long for yesterday-ay-ay-ay.

Yesterday,
The need for back-ups seemed so far away.
I knew my data was all here to stay,
Now I believe in yesterday.

Yesterday, by Colin and the Beatles.

A useful reminder to take regular backups.

16. Languages for RTS dev - C, ADA & Java

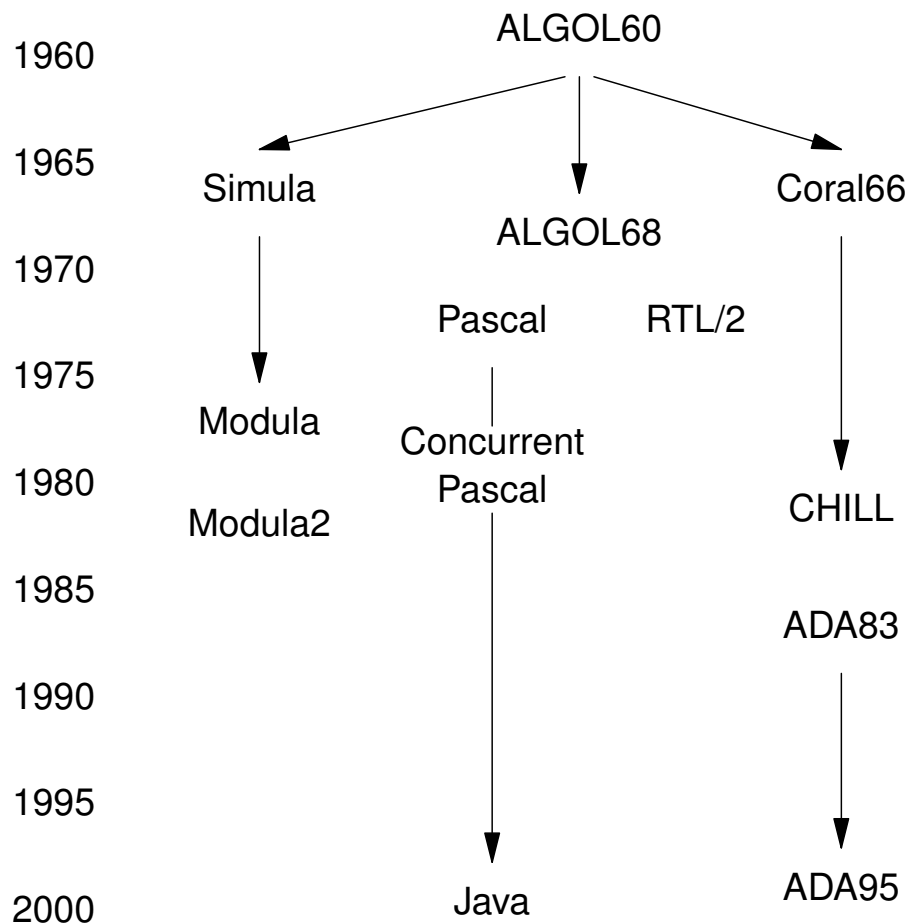


Fig. 16.2 R-t languages

- Readability
- Flexibility
- Portability
- Understandability
- Reliability
- Supportability
- Maturity
- Efficiency

Fig. 16.3a General language needs: all the 'bilities

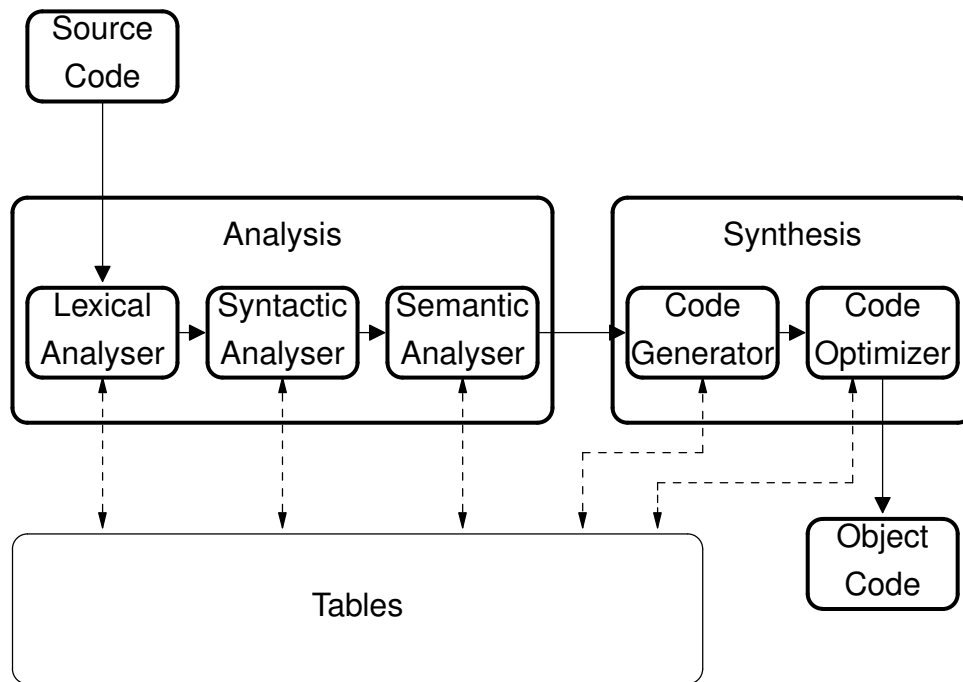


Fig. 16.3b Components of a HLL Compiler

- Multi-tasking facilities
- Re-entrant code generated
- Initialization source code
- Run-time efficiency
- Interrupt handling
- Error recovery
- Predictable performance
- Hardware access
- Linker facilities
- Excellent debugger
- Specialist libraries

Fig. 16.4 R-t compilers.



Fig. 16.5a Speed and dependability

- Elimination of ineffectual code
- Reduction in arithmetic strength
- Identification of common sub-expressions
- Substitution of in-line macros for functions
- Compile time arithmetic
- Removal of loop invariants
- Use of loop index variable
- Registers & caches
- Unused instructions
- Optimization of flow-of-control
- Constant propagation
- Redundant store elimination
- Dead-variable elimination
- Boolean expressions
- Loop unrolling
- Loop amalgamation
- Reduction of conditional switches

Fig. 16.5b Compiler optimization techniques.

```

-falign-functions=n -falign-jumps=n -falign-labels=n
-falign-loops=n -fbranch-probabilities -fcaller-saves
-fcprop-registers -fcse-follow-jumps -fcse-skip-blocks
-fdata-sections -fdelayed-branch -fdelete-null-pointer-checks
-fexpensive-optimizations -ffast-math -ffloat-store -fforce-add
-fforce-mem -ffunction-sections -fgcse -fgcse-lm -fgcse-sm
-floop-optimize -fcrossjumping -fif-conversion -fif-conversion2
-finline-functions -finline-limit=n -fkeep-inline-functions
-fkeep-static-consts -fmerge-constants -fmerge-all-constants
-fmove-all-movables -fnew-ra -fno-branch-count-reg
-fno-default-inline -fno-defer-pop -fno-function-cse
-fno-guess-branch-probability -fno-inline -fno-math-errno
-fno-peephole -fno-peephole2 -funsafe-math-optimizations
-ffinite-math-only -fno-trapping-math -fno-zero-initialized-in-b
-fomit-frame-pointer -foptimize-register-move -foptimize-sibling
-fprefetch-loop-arrays -freduce-all-givs -fregmove
-frename-registers -freorder-blocks -freorder-functions
-frerun-cse-after-loop -frerun-loop-opt -fschedule-insns
-fschedule-insns2 -fno-sched-interblock -fno-sched-spec
-fsched-spec-load -fsched-spec-load-dangerous -fsignaling-nans
-fsingle-precision-constant -fssa -fssa-ccp -fssa-dce
-fstrength-reduce -fstrict-aliasing -ftracer -fthread-jumps
-funroll-all-loops -funroll-loops --param name=value
-O -O0 -O1 -O2 -O3 -Os

```

Fig. 16.5c The gcc command line options for alternative optimizations

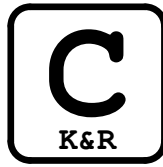


Fig. 16.6 Kernighan & Ritchie's C



Fig. 16.7 Countess Ada Lovelace (1815-52)



Fig. 16.8a Fresh brewed Java

```
public class MyThread extends Thread {
    public void run() {
        while (true) {
            // insert my thread code loop here
        }
    }
}

public class YourThread extends Thread {
    public void run() {
        while (true) {
            // insert your thread code loop here
        }
    }
}

public class TestThreads {
    public static void main( String args[]) {

        MyThread thread_1 = new MyThread();    // create a new thread object
        MyThread thread_2 = new YourThread(); // create a different thread object
        thread_1.start();                       // and run by calling start
        thread_2.start();

    }
}
```

	C	ADA	Java
Cost	free/commodity price	expensive	free/commodity
Readability	programmer dependent notoriously cryptic	good, Pascal-like OOP style possible	C-like OOP style normal
Flexibility	free style	constrained	constrained
Portability	generally very easy, compilers available, bare-board or hosted	few compilers, few BSPs available, usually hosted	first needs JVM ported, then easy, needs native classes to access local h/w
Understandibility	limited syntax expression	modular structure OOP possible	modular structure OOP normal
Reliability	questionable, programmer dependent	exception handling, certified compilers	exception handling
Supportability	good	good	good
Maturity	well established ANSI/ISO	well established ANSI/MIL-STD/ISO	still evolving
Efficiency	fast compiled code	fast compiled code	slower interpreted
Multi-tasking scheduling	programmed with o/s or cyclic executive	built-in tasking facilities	multi-threading
Multi-tasking protection	none	compile time checking	none
Hardware access	normally	possible	deprecated
Interrupt handling	possible with direct h/w access	built-in facilities	none
Safety critical use	doubtful	yes	no
Linker facility for foreign modules	yes	yes	by DLL provision
Init source code available	usually	sometimes	never
asm inserts	yes	yes	no

17. Cross Development Techniques

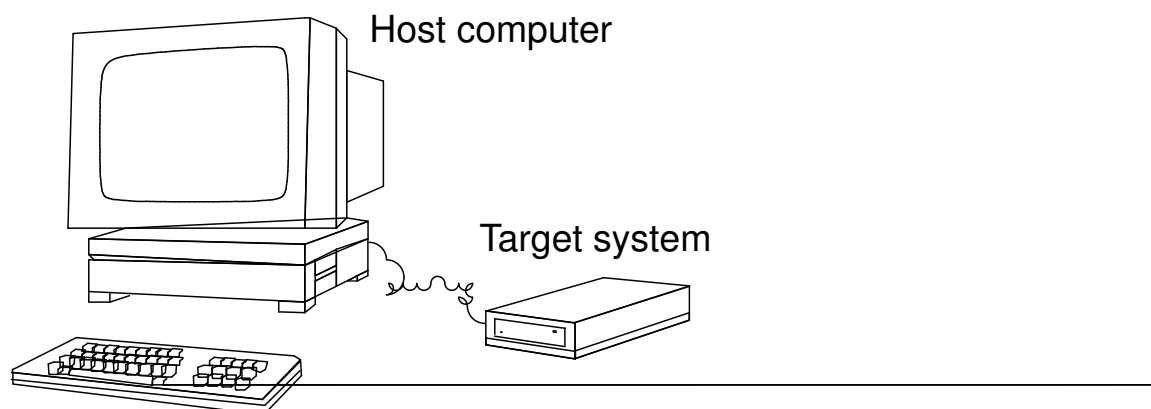


Fig. 17.2 Host-target development



The gcc compiler suite now recognizes the following CPU names:

1750a, a29k, alpha, arm, avr, cn, clipper, dsp16xx, elxsi, fr30,
h8300, hppa1.0, hppa1.1, i370, i386, i486, i586, i686, i786, i860,
i960, m32r, m68000, m68k, m6811, m6812, m88k, mcore, mips, mipsel,
mips64, mips64el, mn10200, mn10300, ns32k, pdp11, powerpc, powerpcle,
romp, rs6000, sh, sparc, sparclite, sparc64, v850, vax, we32k.

While the recognized target operating systems are:

386bsd, aix, acis, amigaos, aos, aout, aux, bosx, bsd, clix,
ctix, cxux, dgux, dynix, ebmon, esix, freebsd, hms, genix,
gnu, linux, linux-gnu, hiux, hpux, iris, irix, isc, luna, lynxos,
mach, minix, msdos, mvs, netbsd, newsos, nindy, ns, osf, osfrose, ptx,
riscix, riscos, rtu, sco, sim, solaris, sunos, sym, sysv, udi, ultrix,
unicos, uniplus, unos, vms, vsta, vxworks, winnt, xenix.

-nostartfiles	no default startup files
-nostdlib	no default libraries
-Xlinker option -Xlinker param	} Pass the option and parameter directly to the linker
-Wl, option param	
-v	verbose display
-fpic	generate position-independent code

Fig. 17.4 Useful gcc options for embedded systems developers



Fig. 17.5a Start-up arrangements

- Initialize any h/w that still needs to be initialized.
- Set up stack and heap
- Clear the .bss section for uninitialized data
- Copy initial values from .text to .data
- Handle `argc` and `argv[]` command line arguments
- Transfer from `crt0`, startup code, to `main()`
- Cleanup After `main()` has completed

Fig 17.5b Start up code

```

rts-rob@kenny> gcc -v world.c
Reading specs from /usr/lib/gcc-lib/i386-linux/3.3/specs
Configured with: ../src/configure -v --enable-languages=c,c++,java,f77,pascal,
    objc,ada,treelang
--prefix=/usr
--mandir=/usr/share/man
--infodir=/usr/share/info
--with-gxx-include-dir=/usr/include/c++/3.3
--enable-shared
--with-system-zlib
--enable-nls
--without-included-gettext
--enable-__cxa_atexit
--enable-clocale=gnu
--enable-debug
--enable-java-gc=boehm
--enable-java-awt=xlib
--enable-objc-gc i386-linux
Thread model: posix
gcc version 3.3 (Debian)
    /usr/lib/gcc-lib/i386-linux/3.3/cc1 -quiet -v -D__GNUC__=3 -D__GNUC_MINOR__=3
    -D__GNUC_PATCHLEVEL__=0 world.c -quiet -dumpbase world.c -auxbase world
    -version -o /tmp/cc1Rmoi.s
GNU C version 3.3 (Debian) (i386-linux)
    compiled by GNU C version 3.3 (Debian).
GGC heuristics: --param ggc-min-expand=100 --param ggc-min-heapsize=131072
ignoring nonexistent directory "/usr/i386-linux/include"
#include "..." search starts here:
#include <...> search starts here:
    /usr/local/include
    /usr/lib/gcc-lib/i386-linux/3.3/include
    /usr/include
End of search list.
    as -V -Qy -o /tmp/ccv94vCj.o /tmp/cc1Rmoi.s
GNU assembler version 2.13.90.0.18 (i386-linux)
using BFD version 2.13.90.0.18 20030121 Debian GNU/Linux
/usr/lib/gcc-lib/i386-linux/3.3/collect2 --eh-frame-hdr -m elf_i386 -dynamic-linker
/lib/ld-linux.so.2
/usr/lib/gcc-lib/i386-linux/3.3/../../../../crt1.o
/usr/lib/gcc-lib/i386-linux/3.3/../../../../crti.o
/usr/lib/gcc-lib/i386-linux/3.3/crtbegin.o
-L/usr/lib/gcc-lib/i386-linux/3.3
-L/usr/lib/gcc-lib/i386-linux/3.3/../../../../
/tmp/ccv94vCj.o
-lgcc -lgcc_eh -lc -lgcc -lgcc_eh
/usr/lib/gcc-lib/i386-linux/3.3/crtend.o
/usr/lib/gcc-lib/i386-linux/3.3/../../../../crtfn.o

```

Fig 17.5c The gcc compiler -v flag

```

> gcc -c testprog.c
> ld -T testprog.lds
>
STARTUP(crt0.o)                /* file for first position in link list */
OUTPUT_ARCH(arm)
INPUT(libc.a libg.a testprog.o) /* input file list for linking */
OUTPUT_FORMAT("arm-elf")
OUTPUT_FILENAME("testprog")    /* can be overridden by command line */
SEARCH_DIR(.)
ENTRY(_start)                  /* ignored by reset vector */
__DYNAMIC = 0;
MEMORY {                       /* target memory map */
    vect (RX) : ORIGIN = 0x000000, LENGTH = 0.25K
    rom (RX)  : ORIGIN = 0x000000, LENGTH = 16M
    ram (WX)  : ORIGIN = 0x1000000, LENGTH = 32M
}
SECTION {                      /* setup RAM map */
    .vect : {
        __vect_start = .;
        *(.vect);
        __vect_end = .;
    } > vect
    .text : {                  /* .text, CODE SECTION */
        CREATE_OBJECT_SYMBOLS
        __text_start = .;
        *(.text) /* insert code sections for files named on the ld command line */
        *(.strings) /* char strings from all files on ld command line */
        __text_end = .;
    } > rom
    .bss : {                   /* .bss, UNINITIALIZED DATA SECTION */
        __bss_start = ALIGN(0x8);
        *(.bss) /* bss sections from all command line files */
        *(COMMON)
        __bss_end = .;
    } > ram
    .data : AT(__text_end {    /* .data, PREINITIALIZED DATA SECTION */
        __data_start = .;
        *(.data) /* data sections from all command line files */
        __data_end = .;
    } > ram
    .stack : {
        __stack_start = .;
        *(.stack) /* stack sections from all command line files */
        __stack_end = .;
    } > ram
    .stab.(NOLOAD): {         /* .stab, SYMBOL TABLE SECTION */
        *(.stab)
    }
    .strtab.(NOLOAD) : {      /* .strtab, STRINGS TABLE */
        *(.strtab)
    }
}

```

Fig. 17.6 A gcc linker script

- ld command line `-e entriypoint` option
- use of `ENTRY(entriypoint)` within the linker script
- relying on the `start` symbol
- default to the beginning of the `.text` section
- use the hardware reset to force entry at address 000000

Fig. 17.7 Ways to set the main entry point



Fig. 17.8 Impression of a Gnu

```
rob: > export TARGET=arm-elf
rob: > export PREFIX=/tmp/install
rob: > export PATH=${PREFIX}/bin:${PATH}
rob: > cd /tmp
rob:/tmp> mkdir build-gcc build-binutils build-newlib build-glibc install

rob:/tmp> ftp gcc.gnu.org
Connected to gcc.gnu.org.
220 FTP server ready.
Name (gcc.gnu.org:rwilliam): anonymous
331 Guest login ok, send your complete e-mail address as password.
Password:*****
230 Guest login ok, access restrictions apply.
ftp> binary
200 Type set to I.
ftp> dir
200 PORT command successful.
150 Opening ASCII mode data connection for directory listing.
total 3200
dr-xr-xr-x    2 root    root          4096 Feb 14  2004 bin
dr-xr-xr-x    2 root    root          4096 Feb 14  2004 etc
drwxr-xr-x    2 root    root          4096 Feb 14  2004 lib
-rw-r--r--    1 root    root       1449788 Oct 28 06:00 ls-lR
-rw-r--r--    1 root    root       154770 Oct 28 06:00 ls-lR.gz
drwxrwxr-x   45 root    root          4096 Oct 28 12:07 pub
drwxr-xr-x    3 root    root          4096 Nov  8  1999 usr
drwxr-xr-x    3 root    root          4096 Dec  7  2001 www
226 Transfer complete.
499 bytes received in 0.27 seconds (1.79 Kbytes/s)
```

```
ftp> cd /pub/binutils/releases
```

```
250 CSD command successful
```

```
ftp> get binutils-2.15.tar.gz
```

```
200 PORT command successful.
```

```
150 Opening ASCII mode data connection for binutils-2.150.tar.gz (15134701 b
```

```
ftp> cd /pub/gcc/releases
```

```
250 CSD command successful
```

```
ftp> Bget gcc-3.4.2.tar.bz2
```

```
200 PORT command successful.
```

```
150 Opening ASCII mode data connection for gcc-3.4.2.tar.bz2 (27246826 bytes
```

```
ftp> cd /pub/glibc/releases
```

```
250 CSD command successful
```

```
ftp> get glibc-2.3.3.tar.gz
```

```
200 PORT command successful.
```

```
150 Opening ASCII mode data connection for glibc-2.3.3.tar.gz (17489958 byte
```

```
ftp> quit
```

```
rob:/tmp> ftp sources.redhat.com
```

```
Connected to redhat.com
```

```
220 FTP server ready.
```

```
Name (redhat.com:rwilliam): anonymous
```

```
331 Guest login ok, send your complete e-mail address as password.
```

```
Password:*****
```

```
230 Guest login ok, access restrictions apply.
```

```
ftp> binary
```

```
200 Type set to I.
```

```
ftp> dir
```

```
200 PORT command successful.
```

```
150 Opening ASCII mode data connection for directory listing.
```

```
total 3200
```

dr-xr-xr-x	2	root	root	4096	Feb 14	2004	bin
dr-xr-xr-x	2	root	root	4096	Feb 14	2004	etc
drwxr-xr-x	2	root	root	4096	Feb 14	2004	lib
-rw-r--r--	1	root	root	1449788	Oct 28	06:00	ls-lR
-rw-r--r--	1	root	root	154770	Oct 28	06:00	ls-lR.gz
drwxrwxr-x	45	root	root	4096	Oct 28	12:07	pub
drwxr-xr-x	3	root	root	4096	Nov 8	1999	usr
drwxr-xr-x	3	root	root	4096	Dec 7	2001	www

```
226 Transfer complete.
```

```
499 bytes received in 0.27 seconds (1.79 Kbytes/s)
```

```
ftp> cd /pub/newlib/
```

```
250 CSD command successful
```

```
ftp> get newlib-1.12.0.tar.gz
```

```
200 PORT command successful.
```

```
150 Opening ASCII mode data connection for newlib-1.12.0.tar.gz (6684150 by
```

```
ftp> quit
```

Now to unpack the compressed tar files. This will create four new directories: /tmp/gcc-3.4.2, /tmp/binutils-2.15, /tmp/newlib-1.12.0 and /tmp/glibc-2.3.3

```
rob:/tmp> tar -jvxf gcc-3.4.2.tar.bz2
rob:/tmp> tar -zvxvf binutils-2.15.tar.gz
rob:/tmp> tar -zvxvf newlib-1.12.0.tar.gz
rob:/tmp> tar -zvxvf glibc-2.3.3.tar.gz

rob:/tmp> cd build-binutils
rob:/tmp/build-binutils> ../binutils-2.15/configure \
?   --target=$TARGET --prefix=$PREFIX
rob:/tmp/build-binutils> make all install 2>&1 |tee binutils_make.logfile
rob:/tmp/build-binutils> cd ..
rob:/tmp>
```

```
rob:/tmp> cd build-gcc
rob:/tmp/build-gcc> ../gcc-3.4.2/configure --target=$TARGET --prefix=$PREFIX \
?   --with-gnu-ld --with-gnu-as --enable-languages=c \
?   --with-newlib --without-headers --disable-shared
rob:/tmp/build-gcc> make all-gcc install-gcc 2>&1|tee gcc_make.logfile
rob:/tmp/build-gcc> cd ..
rob:/tmp>
```

At this point there will be a preliminary 'bootstrap' compiler, arm-elf-gcc, in \${PREFIX}/bin which can be used to build a target versions of newlib and associated header files.

```
rob:/tmp> cd build-newlib
rob:/tmp/build-newlib> ../newlib-1.12.0/configure --target=$TARGET --prefix=$PREFIX
rob:/tmp/build-newlib> make all install 2>&1|tee newlib_make.logfile
rob:/tmp/build-newlib> cd ..R
rob:/tmp>
```

In \${PREFIX} will be the static libraries: libc.a libg.a and libm.a. At last the full cross-compiler can be built as follows:

```
rob:/tmp> cd build-gcc
rob:/tmp/build-gcc> rm -rf *
rob:/tmp/build-gcc> ../gcc-3.4.2/configure --target=$TARGET --prefix=$PREFIX \
?   --with-gnu-as --with-gnu-ld --enable-languages=c,c++
rob:/tmp/build-gcc> make all install 2>&1|tee make.logfile
rob:/tmp/build-gcc> cd ..
rob:/tmp>
```

Linking view	Execution view
ELF header	ELF header
Program header table (optional)	Program header table
Section 1	Segment 1
Section 2	
Section N	Segment N
Section header table	Section header table (optional)

Fig. 17.10a ELF file format views

Section	Content
.text	program code and constant data
.data	initialized data items
.bss	uninitialized data area
.symtab	module symbol table
.strtab	program strings table
.shstrtab	section names
.rela***	relocation info for module ***

Fig. 17.10b ELF object module sections

```

typedef struct {
    unsigned char    e_ident[EI_NIDENT]; /* 16 char identifier */
    Elf32_Half       e_type;               /* file type 1-rel, 2-exe, 3-so, 4-core */
    Elf32_Half       e_machine;            /* CPU ident */
    Elf32_Word       e_version;            /* ELF file version, 0-inval, 1-current */
    Elf32_Addr       e_entry;              /* virtual address entry point */
    Elf32_Off        e_phoff;              /* program header offset */
    Elf32_Off        e_shoff;              /* section header offset */
    Elf32_Word       e_flags;               /* processor-specific flags */
    Elf32_Half       e_ehsize;              /* ELF header size, in bytes */
    Elf32_Half       e_phentsize;          /* prog header's field size in bytes */
    Elf32_Half       e_phnum;              /* prog header number of fields */
    Elf32_Half       e_shentsize;          /* section header size in bytes */
    Elf32_Half       e_shnum;              /* section header number of entries */
    Elf32_Half       e_shstrndx;           /* section name string index */
} Elf32_Ehdr;

typedef struct {
    Elf32_Word       sh_name;               /* indexes into sectn header string table */
    Elf32_Word       sh_type;               /* 1-progbits, 2-symtab, 3-strngtab, 4-rel */
    Elf32_Word       sh_flags;              /* section attributes */
    Elf32_Addr       sh_addr;               /* if loadable, virtual address of sectn */
    Elf32_Off        sh_offset;             /* sctn offset within loadable image */
    Elf32_Word       sh_size;               /* sctn size in bytes */
    Elf32_Word       sh_link;               /* link to index table */
    Elf32_Word       sh_info;
    Elf32_Word       sh_addralign;          /* 0, 2, 4, 8 word alignment */
    Elf32_Word       sh_entsize;           /* entry size in bytes */
} Elf32_Shdr;

```

Fig. 17.10c ELF main and section headers

The following listing is the output from `readelf` for the `a.out` (ELF format) generated by `gcc` from a trivial `hello.c` program.

ELF Header:

```

Magic:      7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
Class:                               ELF32
Data:                               2's complement, little endia
Version:                               1 (current)
OS/ABI:                               UNIX - System V
ABI Version:                           0
Type:                               EXEC (Executable file)
Machine:                               Intel 80386
Version:                               0x1
Entry point address:                 0x80482a0
Start of program headers:             52 (bytes into file)
Start of section headers:             7452 (bytes into file)
Flags:                               0x0
Size of this header:                  52 (bytes)
Size of program headers:              32 (bytes)
Number of program headers:            7
Size of section headers:              40 (bytes)
Number of section headers:            33
Section header string table index:    30

```

Section Headers:

[Nr]	Name	Type	Addr	Off	Size	ES	Fl
[0]		NULL	00000000	000000	000000	00	
[1]	.interp	PROGBITS	08048114	000114	000013	00	
[2]	.note.ABI-tag	NOTE	08048128	000128	000020	00	
[3]	.hash	HASH	08048148	000148	000028	04	
[4]	.dynsym	DYNSYM	08048170	000170	000050	10	
[5]	.dynstr	STRTAB	080481c0	0001c0	00004c	00	
[6]	.gnu.version	VERSYM	0804820c	00020c	00000a	02	
[7]	.gnu.version_r	VERNEED	08048218	000218	000020	00	
[8]	.rel.dyn	REL	08048238	000238	000008	08	
[9]	.rel.plt	REL	08048240	000240	000010	08	
[10]	.init	PROGBITS	08048250	000250	000017	00	A
[11]	.plt	PROGBITS	08048268	000268	000030	04	A
[12]	.text	PROGBITS	080482a0	0002a0	0001b4	00	A
[13]	.fini	PROGBITS	08048454	000454	00001a	00	A
[14]	.rodata	PROGBITS	08048470	000470	000015	00	
[15]	.eh_frame	PROGBITS	08048488	000488	000004	00	
[16]	.data	PROGBITS	0804948c	00048c	00000c	00	W
[17]	.dynamic	DYNAMIC	08049498	000498	0000c8	08	W
[18]	.ctors	PROGBITS	08049560	000560	000008	00	W
[19]	.dtors	PROGBITS	08049568	000568	000008	00	W
[20]	.jcr	PROGBITS	08049570	000570	000004	00	W
[21]	.got	PROGBITS	08049574	000574	000018	04	W
[22]	.bss	NOBITS	0804958c	00058c	000004	00	W
[23]	.comment	PROGBITS	00000000	00058c	00012c	00	
[24]	.debug_aranges	PROGBITS	00000000	0006b8	000078	00	
[25]	.debug_pubnames	PROGBITS	00000000	000730	000025	00	
[26]	.debug_info	PROGBITS	00000000	000755	000a4f	00	
[27]	.debug_abbrev	PROGBITS	00000000	0011a4	000138	00	
[28]	.debug_line	PROGBITS	00000000	0012dc	000272	00	
[29]	.debug_str	PROGBITS	00000000	00154e	0006af	01	M
[30]	.shstrtab	STRTAB	00000000	001bfd	00011e	00	
[31]	.symtab	SYMTAB	00000000	002244	0006a0	10	
[32]	.strtab	STRTAB	00000000	0028e4	0003ad	00	

Key to Flags:

W (write), A (alloc), X (execute), M (merge), S (strings)

I (info), L (link order), G (group), x (unknown)

O (extra OS processing required) o (OS specific), p (processor spec

continued ->

Program Headers:

Type	Offset	VirtAddr	PhysAddr	FileSiz	MemSiz	Flg	A
PHDR	0x000034	0x08048034	0x08048034	0x000e0	0x000e0	R E	0
INTERP	0x000114	0x08048114	0x08048114	0x00013	0x00013	R	0
[Requesting program interpreter: /lib/ld-linux.so.2]							
LOAD	0x000000	0x08048000	0x08048000	0x0048c	0x0048c	R E	0
LOAD	0x00048c	0x0804948c	0x0804948c	0x00100	0x00104	RW	0
DYNAMIC	0x000498	0x08049498	0x08049498	0x000c8	0x000c8	RW	0
NOTE	0x000128	0x08048128	0x08048128	0x00020	0x00020	R	0
STACK	0x000000	0x00000000	0x00000000	0x00000	0x00000	RW	0

Section to Segment mapping:

Segment Sections...

00	
01	.interp
02	.interp .note.ABI-tag .hash .dynsym .dynstr .gnu.version .gnu.version_r .rel.dyn .rel.plt .init .plt .text .fini .rodata .eh_frame
03	.data .dynamic .ctors .dtors .jcr .got .bss
04	.dynamic
05	.note.ABI-tag
06	

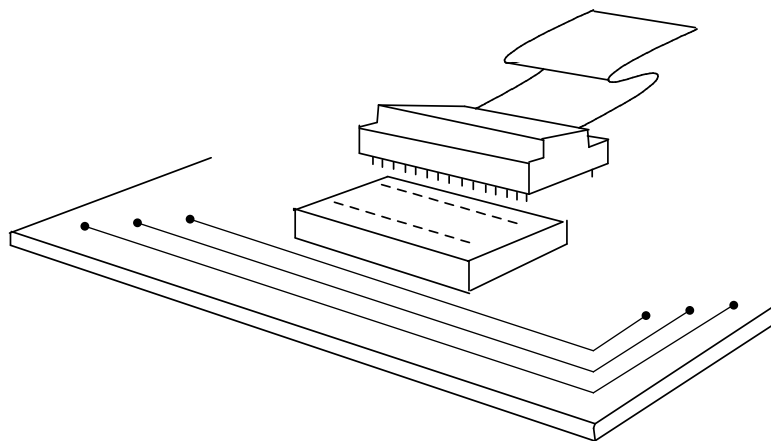


Fig. 17.12a ICE probe and socket

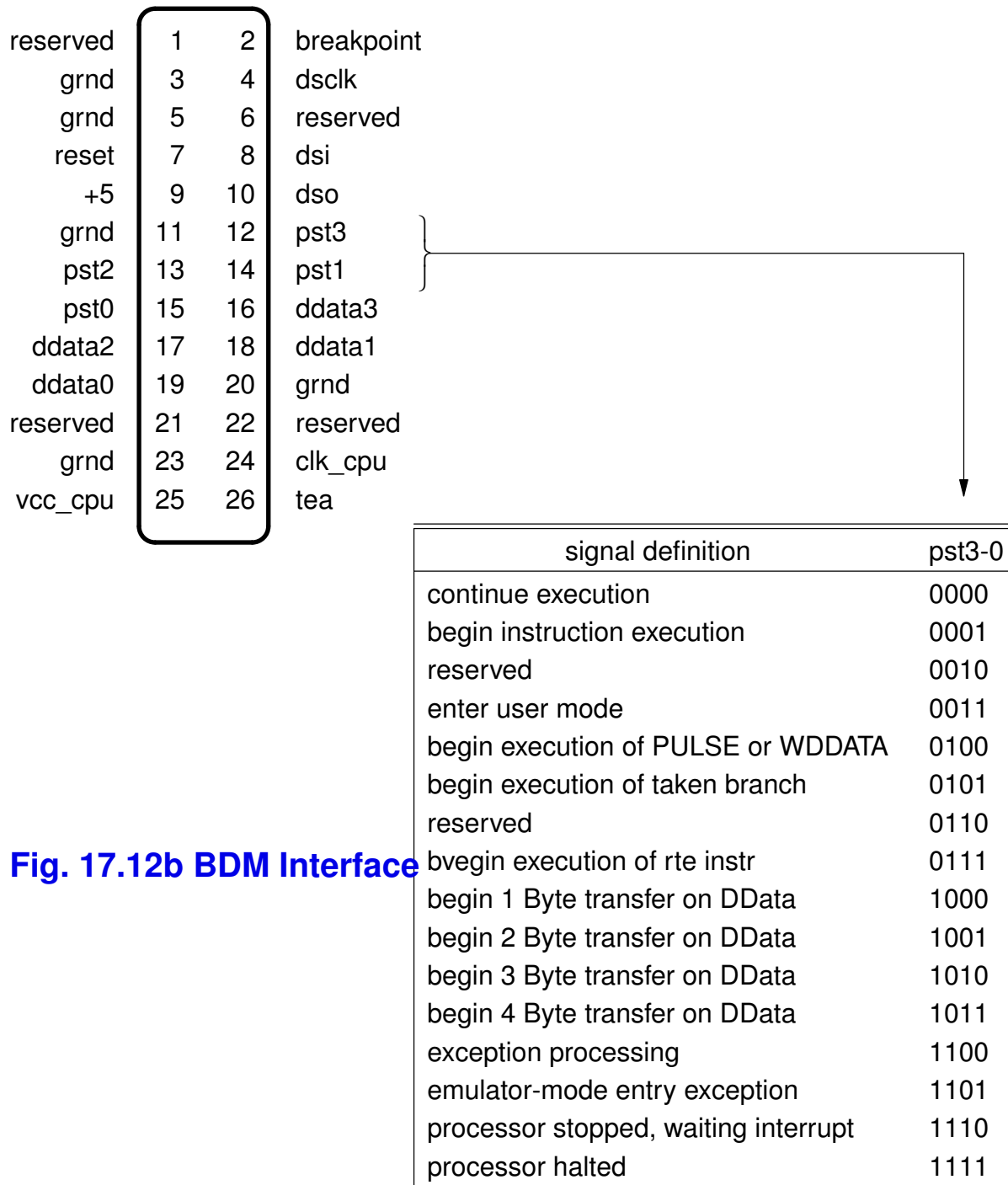


Fig. 17.12b BDM Interface

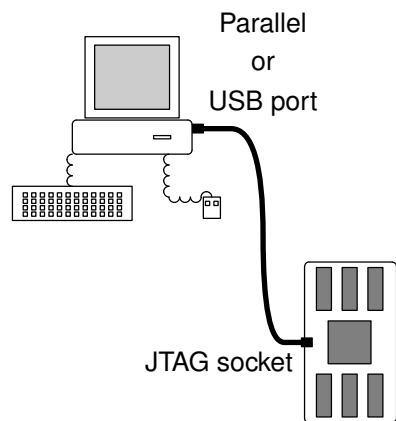


Fig. 17.12c JTAG connection between host and target

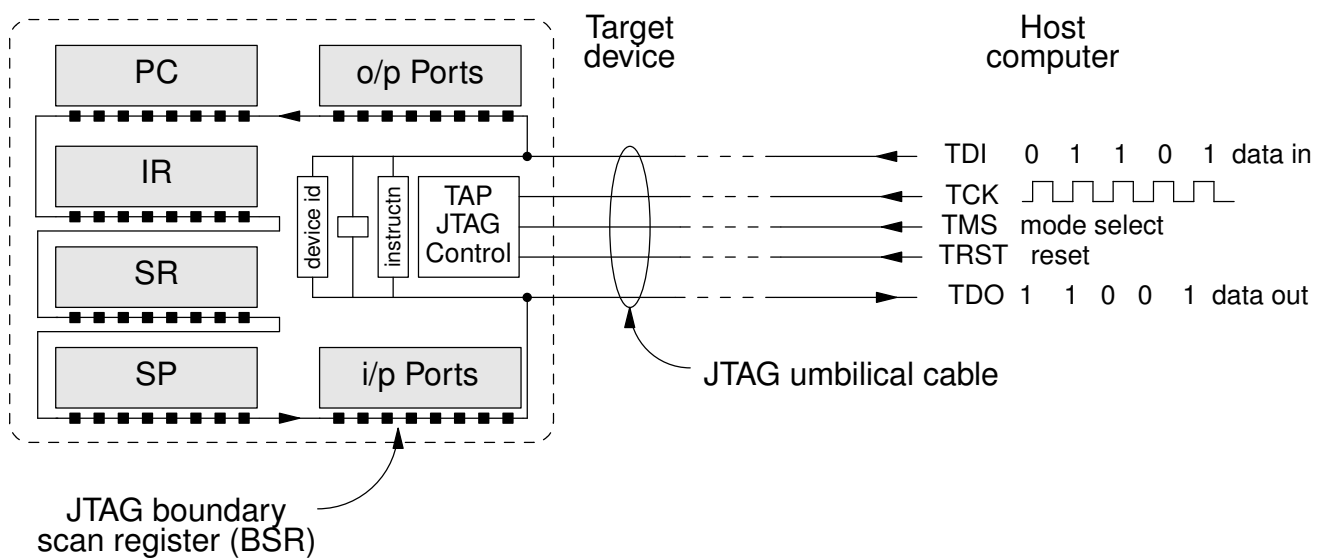


Fig. 17.12d JTAG loop accessing CPU registers

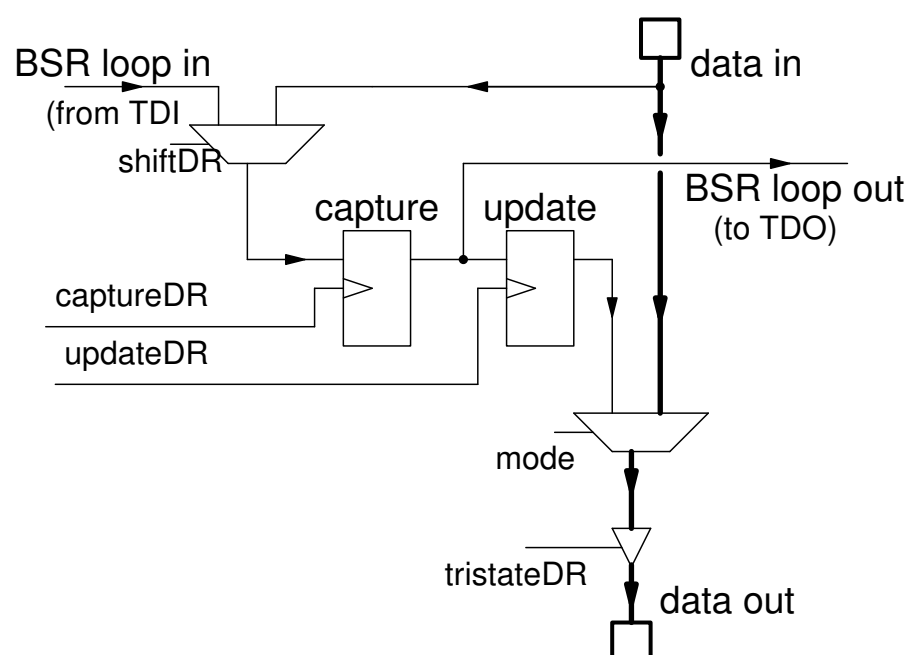


Fig. 17.12e A Boundary Scan Register data cell

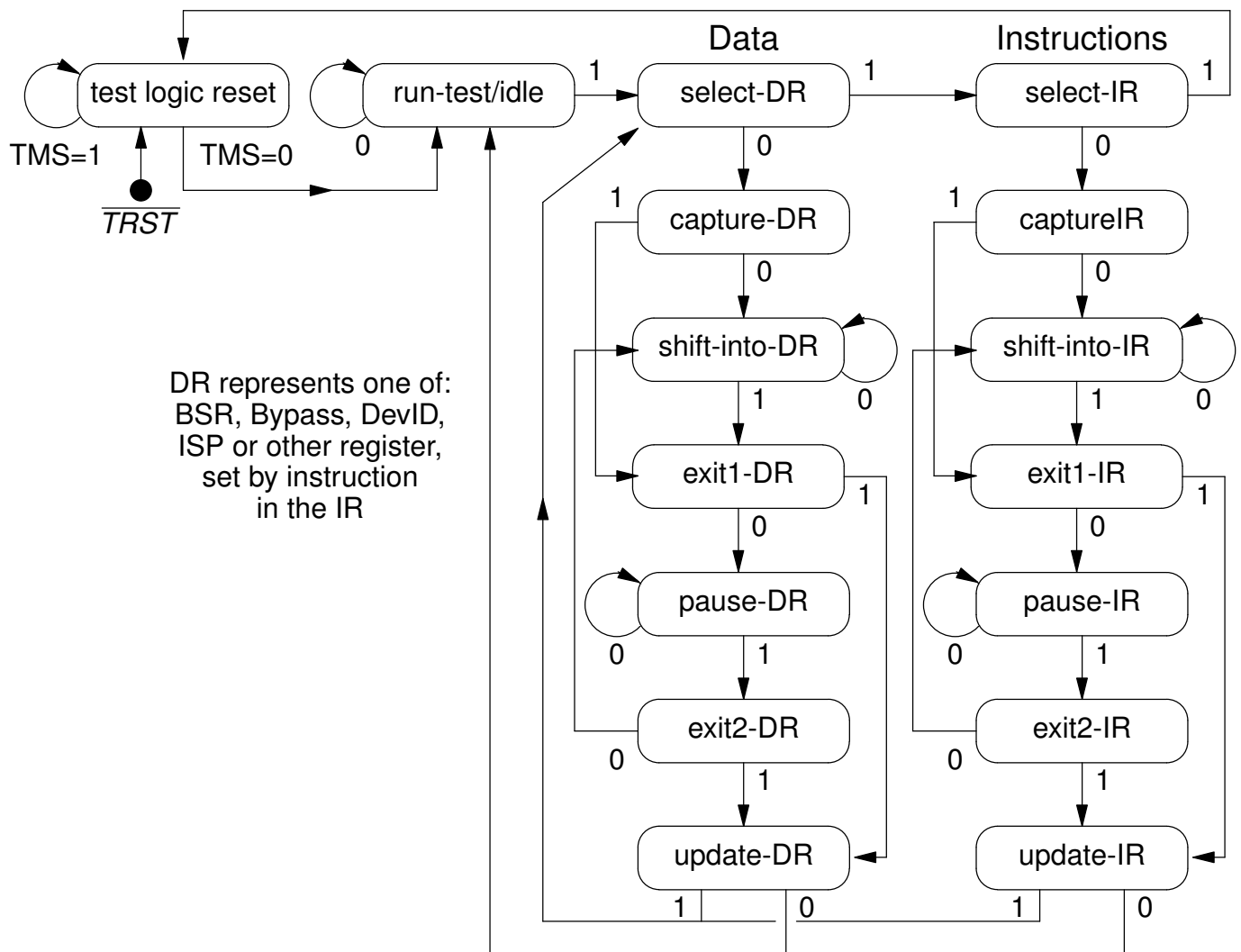


Fig. 17.12f FSD for a TAP JTAG controller

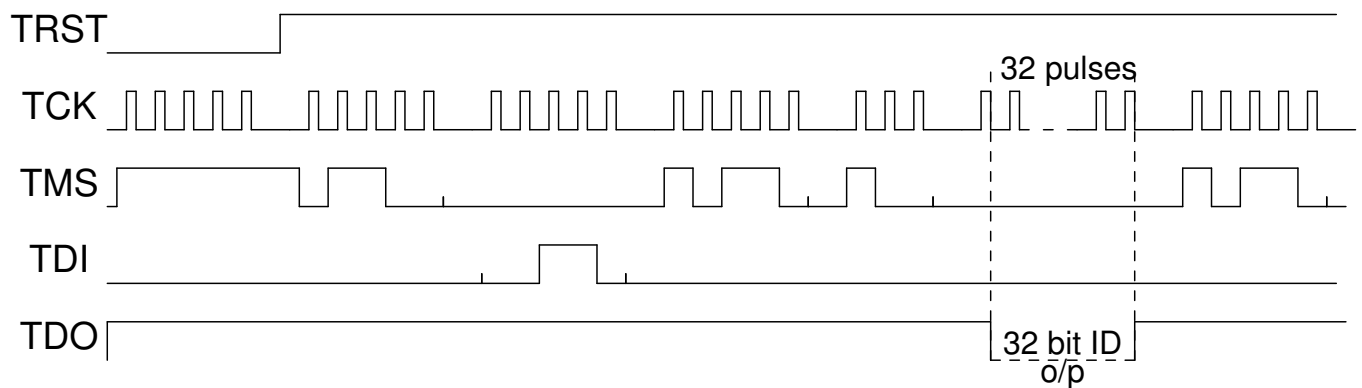


Fig. 17.12g JTAG logic sequence to read a chip ID

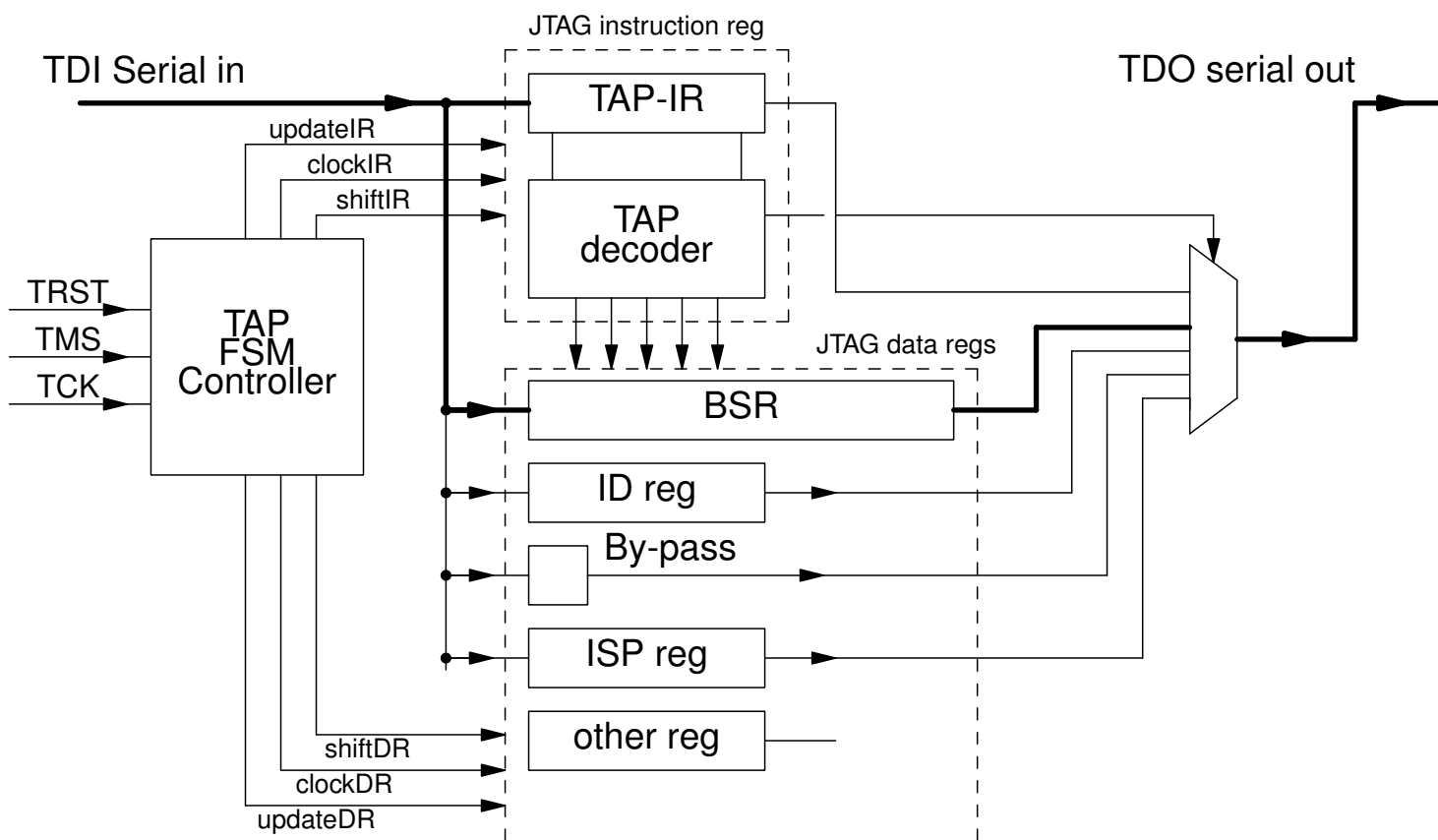


Fig. 17.12h Data paths through the JTAG circuitry

Instruction	Code	
EXTEST	0000	BSR is connected between TDI and TDO, cells latch incoming pin data on entering CAPTURE_DR state, and it gets passed on when exiting from SHIFT_DR state. New contents of BSR are applied to output pins during UPDATE_DR state. Used to test the pack's solder connections.
INTEST	0001	BSR is connected between TDI and TDO, cells latch outgoing core data on entering CAPTURE_DR state, and it gets passed on when exiting from SHIFT_DR state. New contents of BSR are applied to the core logic during UPDATE_DR state. Used to exercise the core without affecting the board connections.
IDCODE	00110	A device id code can be sift out of the id register
BYPASS	11111	Select the single cell bypass, TDI to TDO

Fig. 17.12i Basic set of JTAG instruction codes

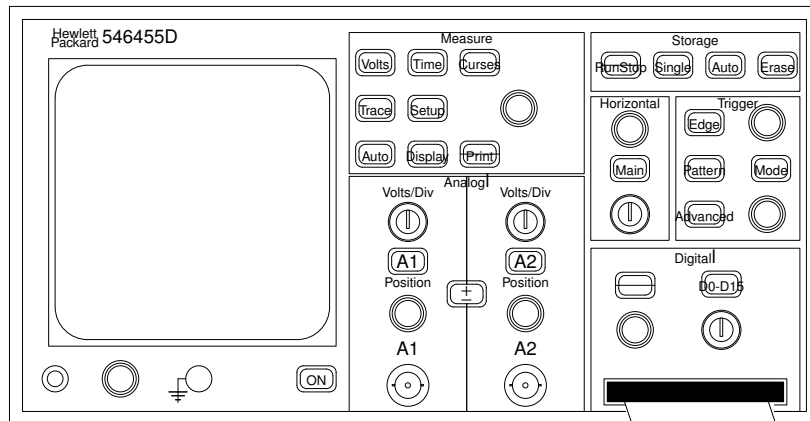


Fig. 17.12j Multichannel Logic Analyser with umbilical connectors

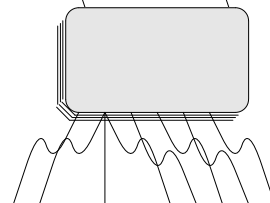
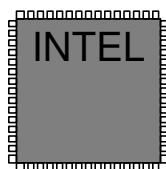


Fig. 17.14 Flash devices

- Operating system disk file
 - USB dongle, local file
 - LAN/NFS, disk file
 - native development
- RTE
 - floppy disk, module file
 - serial port, module file
 - LAN/FTP, module file
- Monitor serial line, hex file
 - hex code from keypad
- JTAG connector serialised binary
- PROM socket executable program

Fig. 17.15a Methods of loading code

$$tottime = \frac{(no_of_words * bus_cycles/write * (lenBSR + updateCycles))}{clockHz}$$

For a 16 KByte segment in FLASH, accessed by 16 bit word; using a host chip with a BSR of 224 cells; and a TCK capable of 100 KHz through the parallel port:

$$totaltime = \frac{(8192 * 4 * (224 + 5))}{100000} = 75 \text{ secs}$$

2 bytes	2 bytes	6 bytes	< 256 bytes	2 bytes
Type	Length	Address	Data	Checksum

Fig. 17.15b Motorola S record format with trailing checksum bytes

```

S0 03 0000 FC
S2 24 010400 46FC26002E7C000808006100005E610000826100033C46FC270023FC0001067
S2 24 010415 000C011023FC00010678000C011423FC00010678000C23FC00010678000C 6D
S2 24 010440 011C610003A4303C271053406600FFFC46FC21006100057A4E4B000000004E7
.....
S2 24 012200 0968584F4878004C4EB900010928584F206EFFF524810BC000460224879000
S2 24 012220 21CA4EB900010968584F487800484EB900010928584F206EFFF524842104E5
S2 08 012240 4E750000 D1
S8 04 000000 FB

```

Fig. 17.15c Motorola S Record file format

08 + 01 + 22 + 40 + 4E + 75 + 00 + 00 = 12E

forget the 1 as overflow, leaving 2E (0010 1110)

invert the bits 1101 0001 (**D1**) THE CHECKSUM !

Objcopy can convert a binary executable image into an S-record format file for downloading:

```
rob@kenny> objcopy -O srec file.bin file.sr
```

18. Microcontroller Embedded Systems

Type	Size	Speed	Examples	Roles
Low cost microcntrl High volume	4/8 bits	12 MHz	Intel 8051 Microchip 16F84	domestic white goods, toys, games
General purpose microcontroller	8/16bits	25 MHz 80 MHz	80186 Infineon c166S	technical equipment, automotive applications
High performance graphics support	32 bits	3 GHz	Intel Pentium IBM Power PC	desktop PC
High performance Graphics support Power save mode	32 bits	2.4 GHz	Intel Atom Transmeta Crusoe	Laptop/portable PC,
Good performance Low power	32 bits	400 MHz 200 MHz	Intel XScale StrongARM	handheld equipment, PDA, Pocket PCs, real-time power computing
Top performance wide address range	32/64 bits	800 MHz	Alpha Itanium	network server, database server

Fig. 18.2 Categories of microprocessor/controller

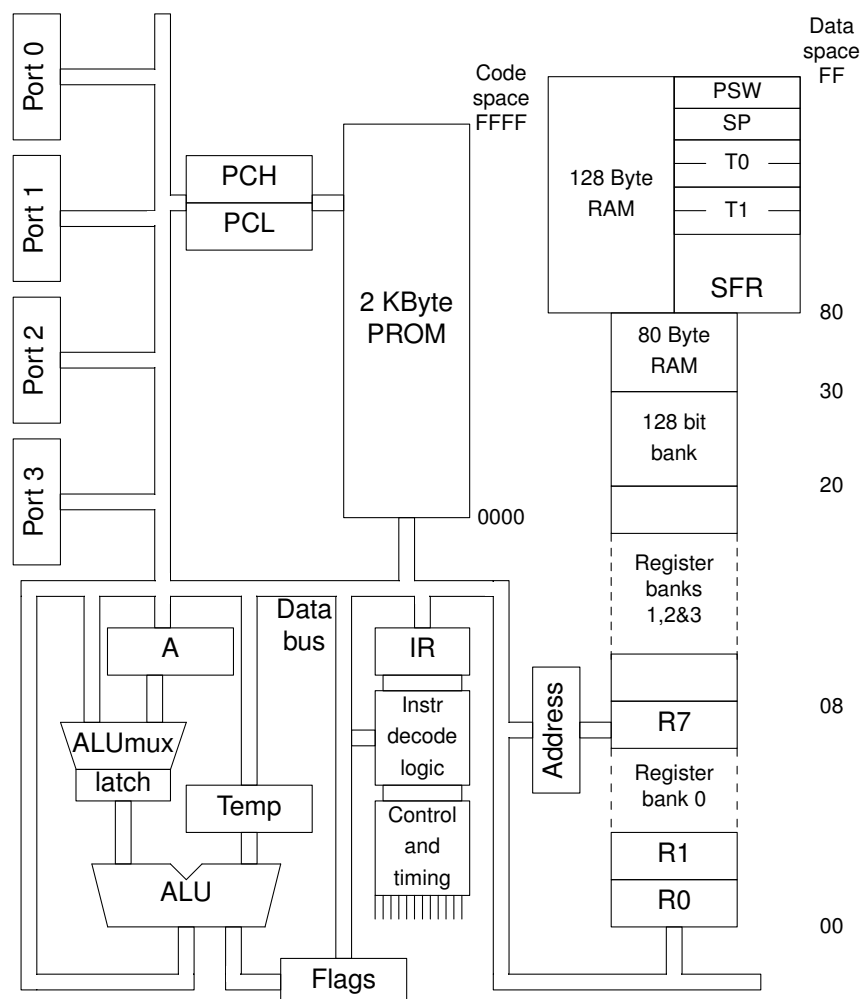


Fig. 18.3a Intel 8051 internal block schematic

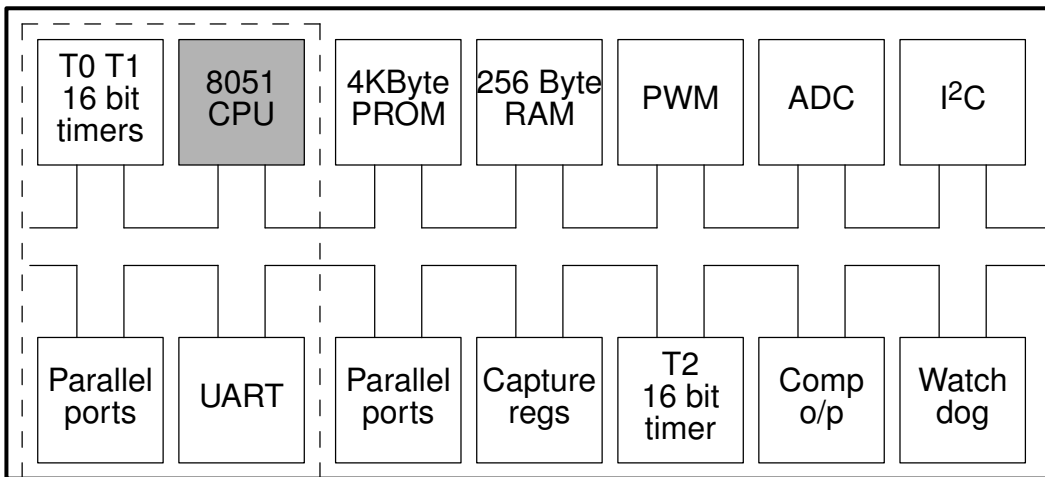


Fig. 18.3b Schematic of Philips PCB83C552 μ controller based on the Intel 8051 core

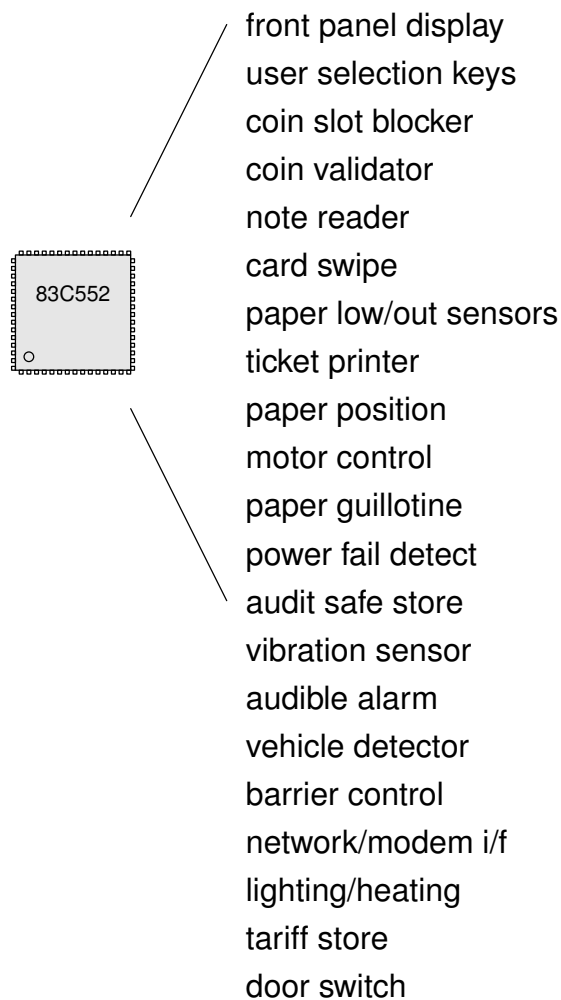


Fig. 18.4a Ticket machine: I/O diversity

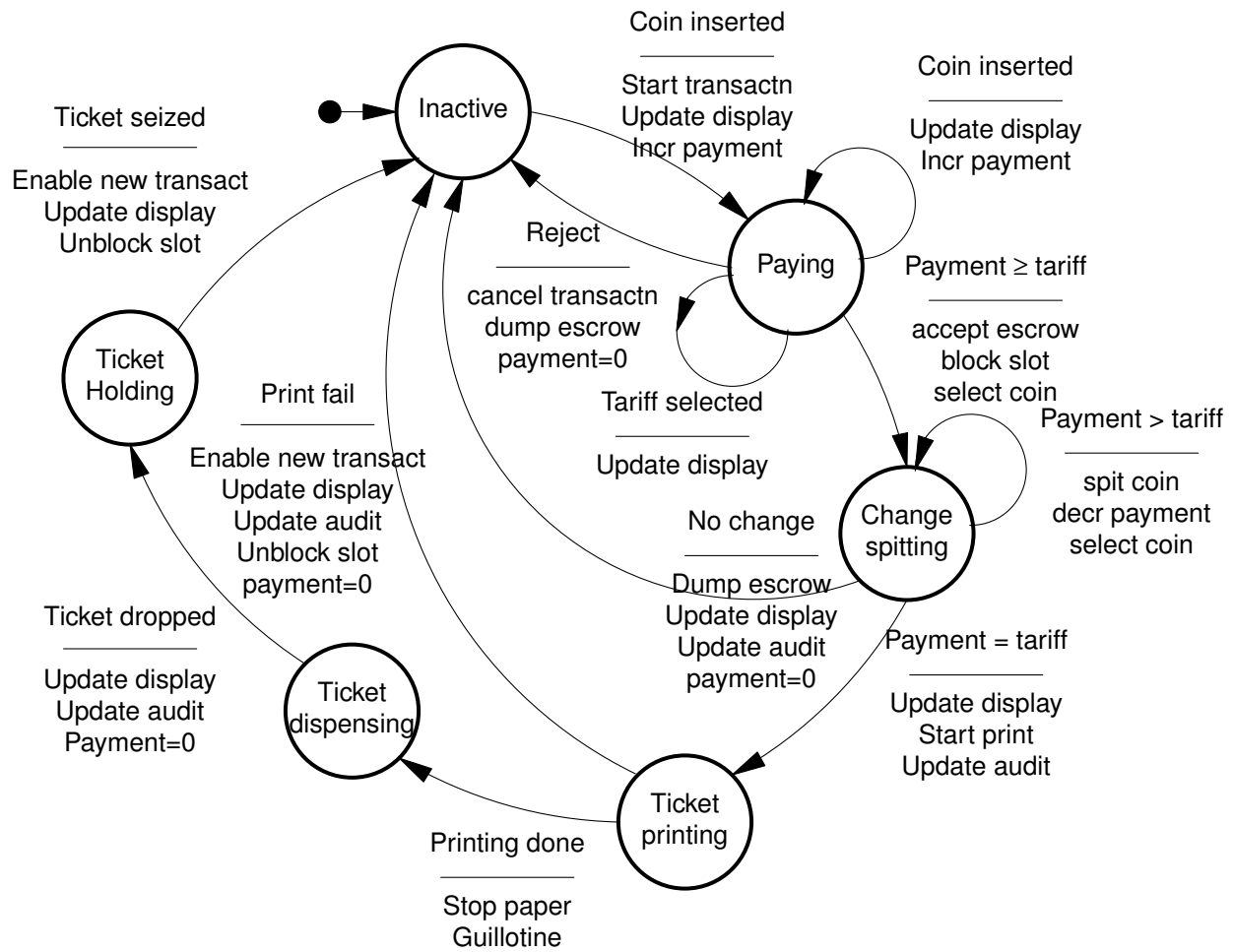


Fig. 18.4b FSD for a simple Pay & Display machine

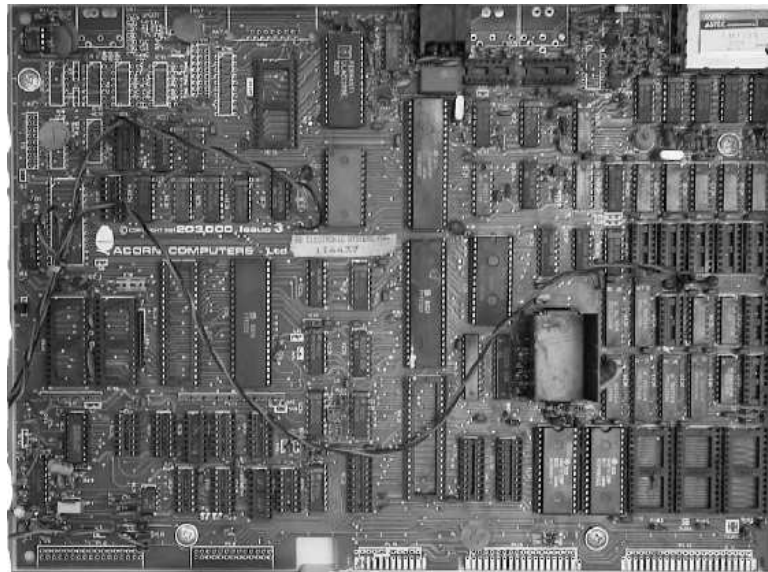


Fig. 18.5 The 1980 BBC Micro motherboard with 6502 processor

Bank	Sel line	MBytes		Physical Address
		384	Reserved	FFFF_FFFFH
3		128	zeros	
3	RAS/CAS3	128	DRAM bank 3	
3	RAS/CAS2	128	DRAM bank 2	
3	RAS/CAS1	128	DRAM bank 1	
3	RAS/CAS0	128	DRAM bank 0	
2		256	LCD & DMA registers	} internal to SA1110 StrongARM
2		256	Expansion & memory	
2		256	SCM registers	
2		256	PM registers	
		768	Reserved	
1	CS5	128	Flash/SRAM bank 5	
1	CS4	128	Flash/SRAM bank 4	
0	PSTSEL	256	PCMIA socket 1	
0	!PSTSEL	256	PCMIA socket 0	
0	CS3	128	Flash/SRAM bank 3	
0	CS2	128	Flash/SRAM bank 2	
0	CS1	128	Flash/SRAM bank 1	
0	CS0	128	Flash bank 0	0000_0000H

Fig. 18.6a SA1110 StrongARM 4 GByte memory map

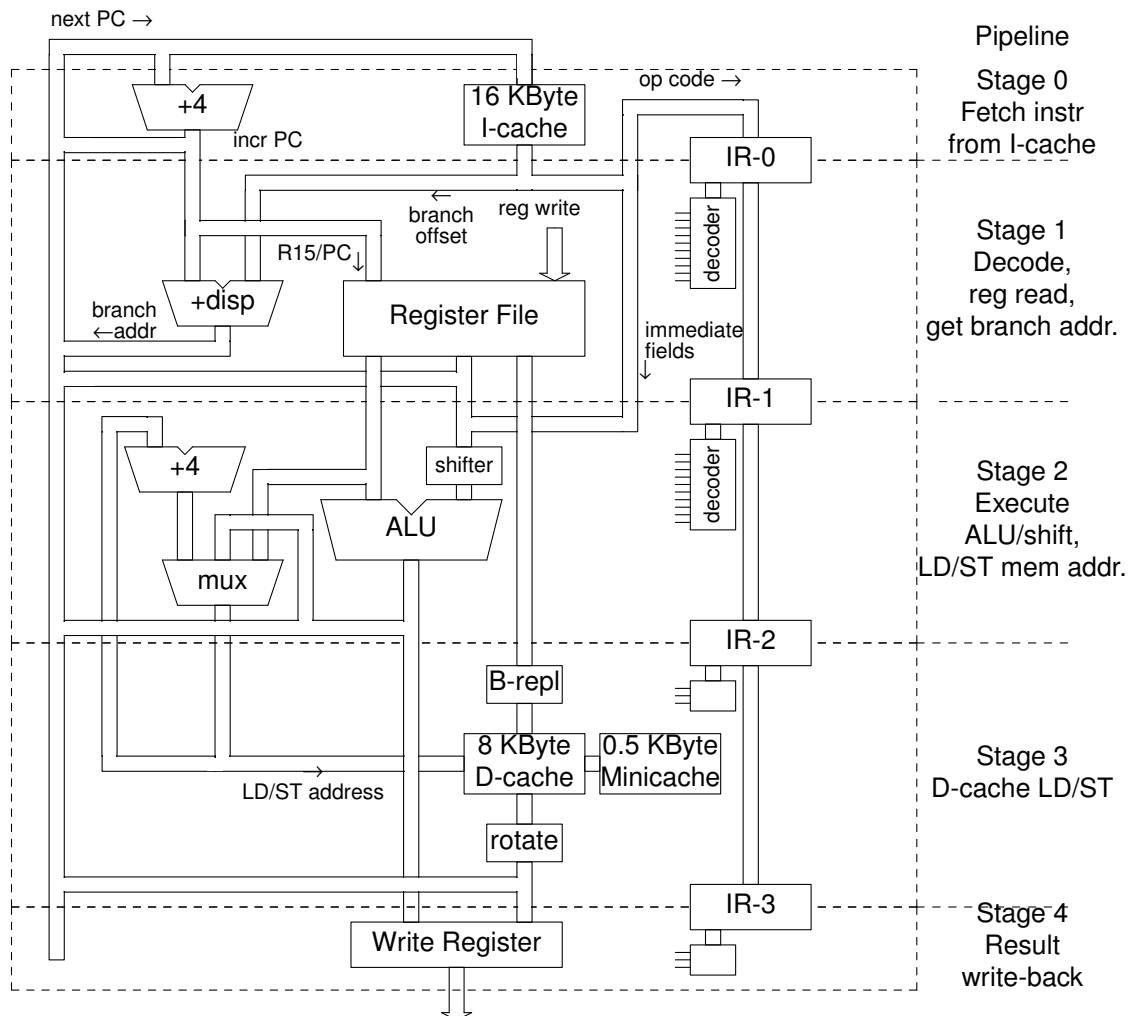


Fig. 18.6b Block diagram for the StrongARM Core

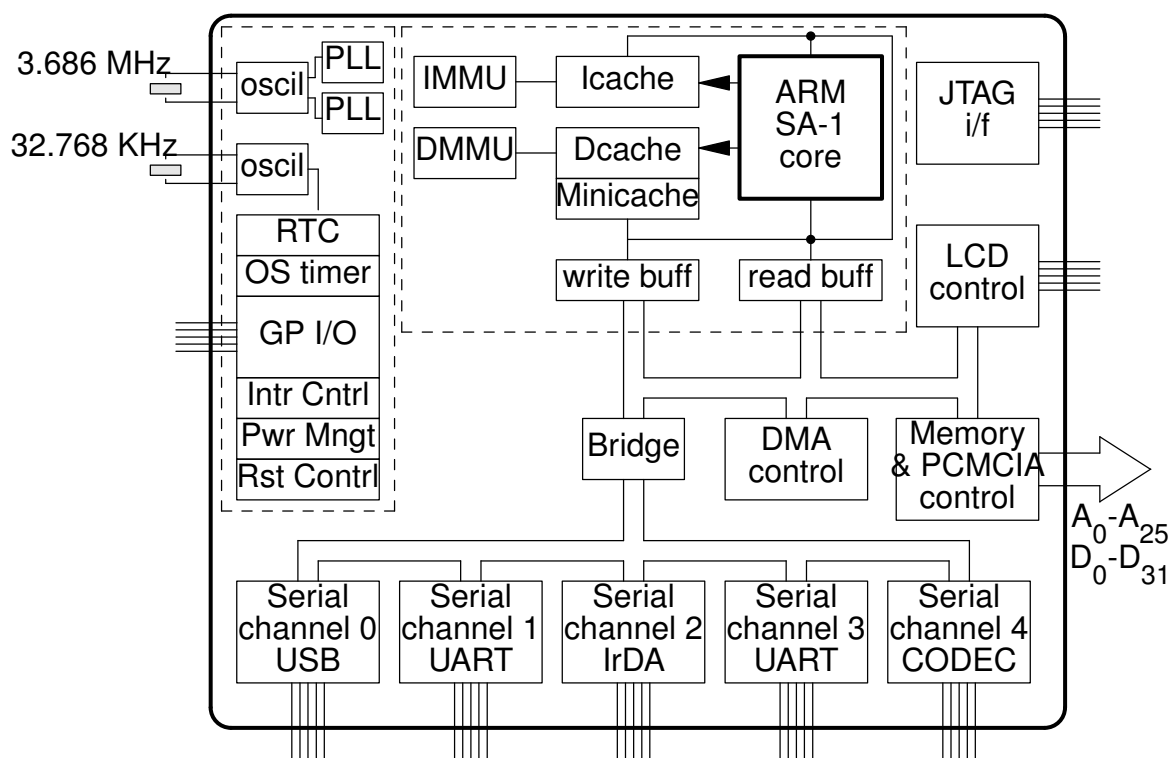


Fig. 18.6c Intel SA1110 StrongARM microcontroller

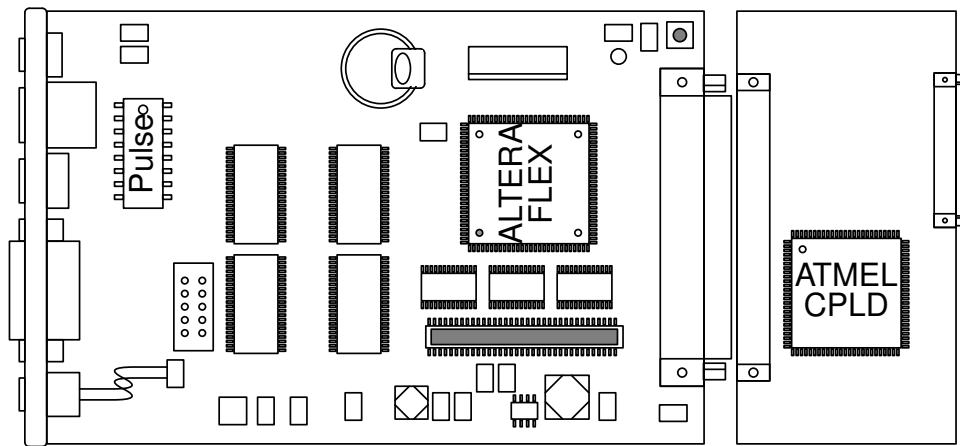


Fig. 18.7a Front side of Puppeteer SBC and a plug in I/O expansion card

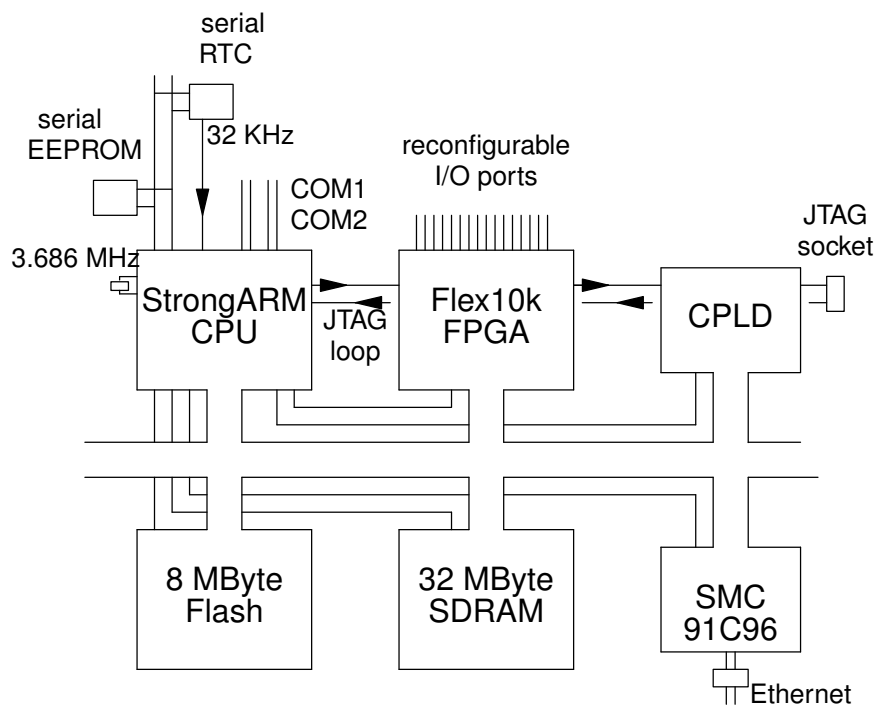


Fig. 18.7b System layout for the StrongARM Puppeteer board

Device range	Size	Physical address	Select line	Virtual address with caching
ZeroBank cache flush	4 MBytes	E000_0000-E03F_FFFF		
SDRAM1	16 MBytes	C800_0000-C8FF_FFFF	RAS1	8100_0000-81FF_FFFF
SDRAM0	16 MBytes	C000_0000-C0FF_FFFF	RAS0	8000_0000-80FF_FFFF
ASB	4 MBytes	B000_0000-B03F_FFFF		
MMU	4 MBytes	A000_0000-A03F_FFFF		
	4 MBytes	9000_0000-903F_FFFF		
Sys Cntrl	4 MBytes	8900_0000-893F_FFFF		
PCMA1	1 MByte	3000_0000-300F_FFFF	PSTsel	
Ethernet	4 MBytes	2000_0300-2000_03FF		
FPGA	4 MBytes	1040_0000-107F_FFFF		
Flash	8 MBytes	0000_0000-007F_FFFF	CS0	9140_0000-91BF_FFFF

Fig. 18.7c Memory map for StrongARM Puppeteer

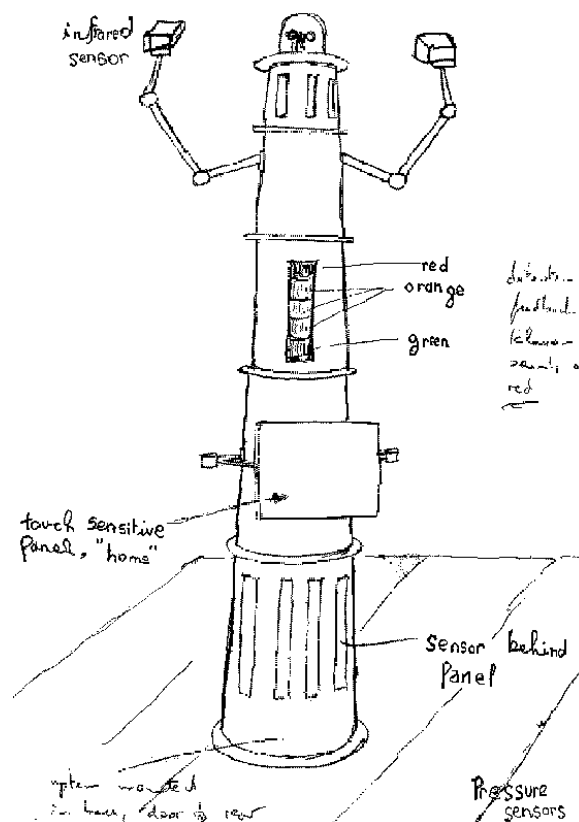


Fig 18.8 The intelligent goal post

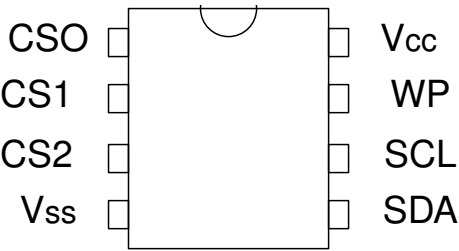


Fig. 18.10a ATMELE Serial EEPROM

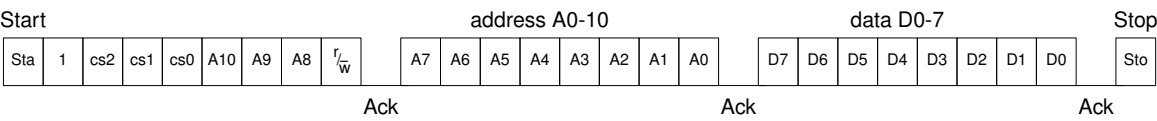


Fig. 18.10b Command sequence for a 1 KByte serial EEPROM

19. Linux Device Drivers

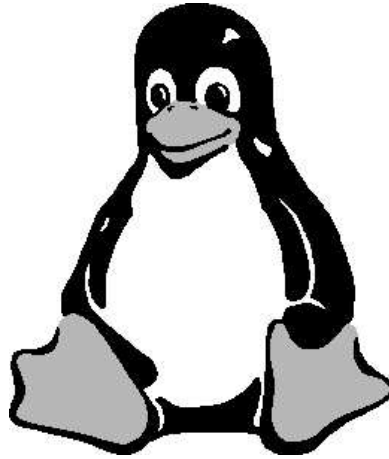


Fig. 19.1 Porting Linux to embedded platforms

```
rob:/tmp> ftp ftp.linux.org
Connected to ftp.linux.org.
220 FTP server ready.
Name (ftp.linux.org:rwilliam): anonymous
331 Guest login ok, send your complete e-mail address as password.
Password: *****
230 Guest login ok, access restrictions apply.
ftp> binary
200 Type set to I.
ftp> dir
200 PORT command successful.
150 Opening ASCII mode data connection for directory listing.
total 3200
dr-xr-xr-x   2 root    root          4096 Feb 14  2004 bin
dr-xr-xr-x   2 root    root          4096 Feb 14  2004 etc
drwxr-xr-x   2 root    root          4096 Feb 14  2004 lib
-rw-r--r--   1 root    root    1449788 Oct 28 06:00 ls-lR
-rw-r--r--   1 root    root    154770 Oct 28 06:00 ls-lR.gz
drwxrwxr-x  45 root    root          4096 Oct 28 12:07 pub
drwxr-xr-x   3 root    root          4096 Nov  8  1999 usr
drwxr-xr-x   3 root    root          4096 Dec  7  2001 www
226 Transfer complete.
499 bytes received in 0.27 seconds (1.79 Kbytes/s)
ftp> cd /pub/linux/releases/2.4
250 CSD command successful
ftp> get linux-2.4.18.tar.gz
200 PORT command successful.
150 Opening ASCII mode data connection for linux.2.14.tar.gz
ftp> quit
```

1. Next the compressed Linux tar file sitting in /tmp has to be decompressed and unpacked:

```
> tar -vzxf linux-2.4.18.tar.gz
```

2. This builds the Linux source file directory tree, with multiple configure and makefiles, based on a new directory /tmp/linux. To adjust the kernel to deal with the target CPU architecture a special patch file has to be obtained and inserted into the source tree. In this case, the ARM patch produced by Russell King is being downloaded, using ftp as before, but from a different ftp server:

```
rob:/tmp> ftp ftp.arm.linux.org.uk
ftp> cd /pub/linux/arm/source/kernel-patches/v2.4
ftp> binary
. . . .
ftp> get patch-2.4.18-rmk7.gz
ftp> quit
```

3. Before applying the architecture patch, it may be necessary to clean up the distribution, perhaps to remove the results of failed earlier attempts.

```
> make distclean
```

4. To apply the architecture amendment patch to the main kernel source:

```
> gzip -cd patch-2.4.18-rmk7.gz | patch -p0
```

This takes the diff format changes held in the patch file, and applies them to the list of files which it also contains. In this case, there are too many files to all be cited on the command line! It is probably worth scanning through `man patch` to understand what effect it is having on the original files, but the `-p` flag indicates whether to truncate the path names by deleting leading /s.

5. Now the top level kernel Makefile, contained in the /tmp/linux directory, needs to be edited appropriately for your set-up:

```
ARCH := ARM
CROSS-COMPILE = arm-linux-
```

6. At this moment, even though the source tree has been adjusted for the target CPU (ARM), the kernel code still needs further adjustment for compilation to work for the target board's memory and I/O devices. One strategy to speed things up, is to look out for an existing configuration which is similar to your new target. Each of the available config files could be inspected in /tmp/linux/arch/arm/ and the best option chosen. In this case, as an example, we will take the Pangolin board as closest.

```
> make pangolin
```

This produces a `.config` file to be used as a starting point for the next make operation, and needs to be placed in `/tmp/linux`.

7. Stay in the `/tmp/linux` directory and run the principal, top-level makefile to configure the kernel. The choice of make targets usually includes: `config`, `xconfig` or `menuconfig`, they all run interactively to allow the correct target system options to be selected. The `xconfig` version is not quite as fully featured as the other two, but perhaps, more convenient.

```
> make menuconfig
```

The response choices offered to each of the config options are Y, N, and ? for more help. The configuration process is driven by the `config.in` text file. While all the information captured is entered into the `.config` file. The `config.in` file is located in `/tmp/linux/arch/arm`, if the target architecture is ARM. If you look inside, the structure and sequence of the `menuconfig` questions can be recognised. More information about the configuration scripting language can be found in the file: `/tmp/linux/Documentation/kbuild/config-language.txt`. The main result of passing through the `make menuconfig` configuration process, is to set the values for variables, or macros, used by the many makefiles held in the Linux source tree. The makefiles will read out these values to determine which components to include in the new kernel, and also to pass `#define` values into various source code files.

8. The next step configures the memory assignments for loading, decompressing and running the kernel. This requires the address values in the `/tmp/linux/arch/arm/boot` file to be edited, as well as a few other memory value revisions to convert to the specific target configuration. Check the address values in the linker script file, `vmlinux.lds`, ensure they are in accord with the target board memory layout for TEXT segment, BSS segment and start addresses.

9. If necessary change back to `/tmp/linux-2.4.18` and then run the makefiles. This make operation is complex and recursive. There are many makefiles scattered over the Linux source tree, so it can be 30 minutes before completion, depending on the speed of your processor.

```
> make dep
> make clean
> make zImage
```

10. Now there should be `vmlinux` in `/tmp/linux/arch/arm/boot/compressed`, this is the compressed kernel with its decompressor code waiting to be downloaded. It can be checked using a bintools utility, `objdump`:

```
> arm-linux-objdump -D vmlinux
```

11. There is also `zImage` in `/tmp/linux/arch/arm/boot`, and `vmlinux` in `/tmp/linux`. But before the compressed kernel boot module can run, a root file system has to be provided for the target system, where it will reside in an 8 MB RAMdisk. The `mkfs` variant command formats the RAMdisk volume for an ext2 file system with 2kB inode size..

```
> dd if=/dev/zero of=imagefile bs=1k count=8192
> /sbin/mke2fs -F -m0 -I 2000 imagefile
> mount -t ext2 -o loop imagefile /loopfs
> cd /loopfs
> mkdir /dev /etc /bin /sbin /lib /usr /proc /tmp /etc/init.d /usr/bin
```

12. Several scripts and files can now be copied into the `/etc` directory from the host file system: `/etc/fstab`, `/etc/init.d/rcs`, `/etc/inittab`, `/etc/passwd`.

13. Also all the necessary special device files with major and minor numbers have to be setup in `/dev` using `mknod`, but this requires root permissions. All the major and minor number values are conventionally standardized and so may be copied from an existing desktop version of Linux.

```
> su
$ mknod -m 0888 tty c 5 0
```

Repeated for: `tty0`, `S0`, `S1`, `SA0`, `SA1`, `SA2`, `cusa0`, `cusa1`, `cusa2`, `ram`, `null`, `pt0`, `pt1`, `pt2`, `pt3`, `ptmx`, `ptyp0`, `ptyp1`, `ptyp2`, `ptyp3`, `ttyp0`, `ttyp1`, `ttyp2`, `ttyp3`.

14. The basic set of Unix tools, `grep`, `sed`, `ls`, etc., can conveniently all be obtained with `busybox`, which can be downloaded from <http://www.busybox.net>.

15. The most suitable library to use with embedded systems might be `uclibc`, which is obtainable from <http://www.uclibc.org> and rebuilt from source.

16. In place of the normal login utility, the smaller `tinylogin` is often preferred for small embedded systems.

17. There remains to figure out how to down-load the compressed kernel module and compressed root file system module into target memory. The best way requires an onboard monitor which can handle ethernet transfers, probably using TFTP. But there are several other alternatives as discussed in Section .

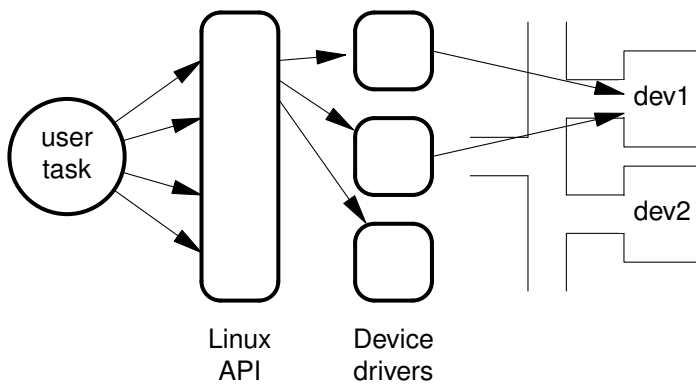


Fig. 19.4 User task access route to devices

```

[rob@local] cat /proc/devices
Character device          Block device
 1 mem                    1 ramdisk
 2 pty/m%d                2 fd
 3 pty/s%d                3 ide0
 4 tts/%d                 9 md
 5 cua/%d                 11 sr
 6 lp                     22 ide1
 7 vcs
10 misc
14 sound
29 fb
116 alsa
128 ptm
129 ptm
130 ptm
131 ptm
136 pts/%d
137 pts/%d
138 pts/%d
139 pts/%d
162 raw
180 usb
[rob@local]
  
```

Fig. 19.5a Major device numbers

```
[rob@local] ls -ls ide/host0/bus0/target*/lun0/part*
0 brw----- 1 root root 3, 1 Jan 1 1970 ide/host0/bus0/target0/lun0/part1
0 brw----- 1 root root 3, 10 Jan 1 1970 ide/host0/bus0/target0/lun0/part10
0 brw----- 1 root root 3, 11 Jan 1 1970 ide/host0/bus0/target0/lun0/part11
0 brw----- 1 root root 3, 2 Jan 1 1970 ide/host0/bus0/target0/lun0/part2
0 brw----- 1 root root 3, 5 Jan 1 1970 ide/host0/bus0/target0/lun0/part5
0 brw----- 1 root root 3, 6 Jan 1 1970 ide/host0/bus0/target0/lun0/part6
0 brw----- 1 root root 3, 7 Jan 1 1970 ide/host0/bus0/target0/lun0/part7
0 brw----- 1 root root 3, 8 Jan 1 1970 ide/host0/bus0/target0/lun0/part8
0 brw----- 1 root root 3, 9 Jan 1 1970 ide/host0/bus0/target0/lun0/part9
0 brw----- 1 root root 3, 65 Jan 1 1970 ide/host0/bus0/target1/lun0/part1
0 brw----- 1 root root 3, 74 Jan 1 1970 ide/host0/bus0/target1/lun0/part10
0 brw----- 1 root root 3, 75 Jan 1 1970 ide/host0/bus0/target1/lun0/part11
0 brw----- 1 root root 3, 66 Jan 1 1970 ide/host0/bus0/target1/lun0/part2
0 brw----- 1 root root 3, 69 Jan 1 1970 ide/host0/bus0/target1/lun0/part5
0 brw----- 1 root root 3, 70 Jan 1 1970 ide/host0/bus0/target1/lun0/part6
0 brw----- 1 root root 3, 71 Jan 1 1970 ide/host0/bus0/target1/lun0/part7
0 brw----- 1 root root 3, 72 Jan 1 1970 ide/host0/bus0/target1/lun0/part8
0 brw----- 1 root root 3, 73 Jan 1 1970 ide/host0/bus0/target1/lun0/part9
[rob@local]          Δ Δ | |
```

Fig. 19.5b Major and Minor device numbers

```
/*
 * Constants relative to the data blocks
 */
#define EXT2_NDIR_BLOCKS 12
#define EXT2_IND_BLOCK EXT2_NDIR_BLOCKS
#define EXT2_DIND_BLOCK (EXT2_IND_BLOCK + 1)
#define EXT2_TIND_BLOCK (EXT2_DIND_BLOCK + 1)
#define EXT2_N_BLOCKS (EXT2_TIND_BLOCK + 1)

/*
 * Structure of an inode on the disk taken from /usr/include/linux/ext2_fs.h
 */
struct ext2_inode {
    __u16    i_mode;      /* File mode */
    __u16    i_uid;      /* Low 16 bits of Owner Uid */
    __u32    i_size;      /* Size in bytes */
    __u32    i_atime;     /* Access time */
    __u32    i_ctime;     /* Creation time */
    __u32    i_mtime;     /* Modification time */
    __u32    i_dtime;     /* Deletion time */
    __u16    i_gid;      /* Low 16 bits of Group Id */
    __u16    i_links_count; /* Links count */
    __u32    i_blocks;    /* Blocks count */
    __u32    i_flags;     /* File flags */
    union { . . . } osd1; /* OS dependent 1 */
    __u32    i_block[EXT2_N_BLOCKS]; /* Pointers to blocks */
    __u32    i_generation; /* File version for NFS */

    /* ACL stuff here */

    union { . . . } osd2; /* OS dependent 2 */
};
```

Fig. 19.6a Unix inode structure on disk

- inode status
 - locked/free
 - task waiting
 - memory inode changed
 - file data changed
 - inode is a mount point
- device number (major & minor)
- inode number
- pointers to other inodes in list
- count of active references

Fig 19.6b Extra data held by in-memory inodes

```

struct stat {
    dev_t      st_dev; /* major-minor device number */
    long       st_pad1[3]; /* reserve for dev expansion, */
    ino_t      st_ino; /* inode number */
    mode_t     st_mode;
    nlink_t    st_nlink; /* number of active links to the file */
    uid_t      st_uid; /* file owner's ID */
    gid_t      st_gid; /* designated group id */
    dev_t      st_rdev;
    long       st_pad2[2];
    off_t      st_size; /* file size in bytes */
    long       st_pad3; /* reserve for future off_t expansion */
    time_t     st_atime; /* last access time */
    time_t     st_mtime; /* last write time (modification) */
    time_t     st_ctime; /* last status change time */
    long       st_blksize;
    long       st_blocks;
    char       st_fstype[_ST_FSTYPSZ];
    long       st_pad4[8]; /* expansion area */
};

```

Fig 19.6c Structure used to access in-memory inode information

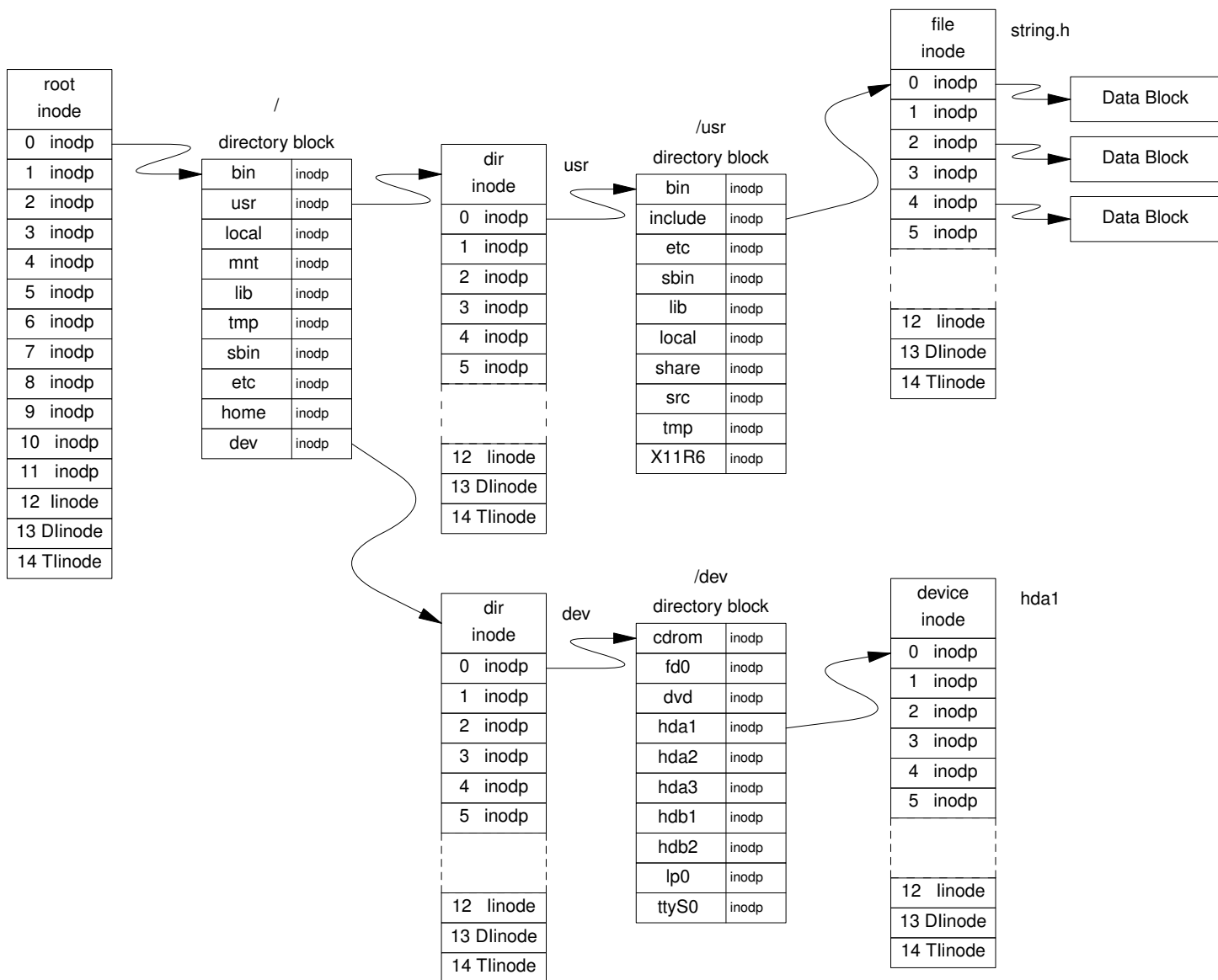


Fig. 19.6d Relating Unix directories, inodes & devices

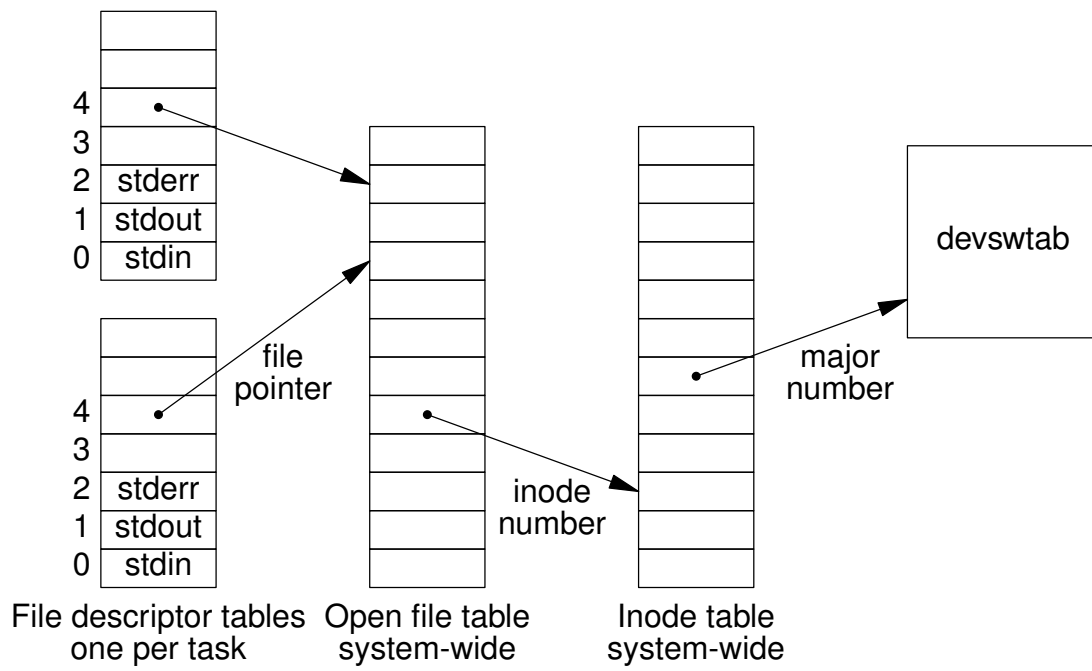


Fig. 19.6e Data structures for open files/devices

```
rob@local> mknod /dev/ttyS0 c 4 64
```

All device have major and minor number, which can be viewed using `ls -l`:

```
rob@local> ls -al /dev/ttyS0
lr-xr-xr-x    1 root    root    5 May 28 19:32 /dev/ttyS0 -> tts/0
rob@local> ls -al /dev/tts/0
crw-rw----    1 rob     tty     4, 64   Jan  1  1970 /dev/tts/0
rob@local>
```

```
[rob@localhost rob] mount
/dev/hda1 on / type ext3 (rw)
none on /proc type proc (rw)
none on /dev type devfs (rw)
none on /dev/pts type devpts (rw,mode=0620)
none on /dev/shm type tmpfs (rw)
/dev/hda8 on /home type ext3 (rw)
/dev/hda5 on /local type ext3 (rw)
/mnt/cdrom on /mnt/cdrom type supermount
(ro,dev=/dev/hdc,fs=iso9660,--,iocharset=iso8859-15)
/mnt/cdrom2 on /mnt/cdrom2 type supermount
(rw,dev=/dev/scd0,fs=iso9660,--,iocharset=iso8859-15)
/mnt/floppy on /mnt/floppy type supermount
(rw,sync,dev=/dev/fd0,fs=vfat,--,iocharset=iso8859-15,umask=0,codepage=850)
/dev/hda10 on /tmp type ext3 (rw)
/dev/hda9 on /usr type ext3 (rw)
/dev/hda6 on /var type ext3 (rw)
none on /proc/bus/usb type usbdevfs (rw,devmode=0664,devgid=43)
/dev/hdb1 on /DOS type vfat (rw)
[rob@localhost rob]
```

```
[rob@localhost rob] df
Filesystem      Size  Used Avail Use% Mounted on
/dev/hda1       5.8G   73M  5.3G   2% /
none            125M    0  124M   0% /dev/shm
/dev/hda8       3.9G  283M   3.6G   8% /home
/dev/hda5       478M   8.1M  445M   2% /local
/dev/hda10      5.1G   34M   4.8G   1% /tmp
/dev/hda9       2.0G  969M  929M  52% /usr
/dev/hda6       1.0G  109M  887M  11% /var
/dev/hdb1       503M  297M  206M  59% /DOS
[rob@localhost rob]
```

- Discrete char devices handling byte streams, sometimes termed *raw* devices, such as:

```
/dev/console      c 5, 1
/dev/ttyS0        c 4, 64serial port
/dev/tty1         c 4, 1terminal
```

- Block data devices, handling 1, 2 or 4kByte blocks of data in single operations.

```
/dev/fd0 b 2, 0    floppy disk
/dev/hda1         b 3, 1IDE hard drive
/dev/sda1         SCSI hard drive
/dev/cd0 b 22,0    CDROM drive
```

- Network interface

```
[rob@localhost rob] cat /procs/kallsyms | wc -l
18328
```

```
[rob@localhost rob] cat /procs/kallsyms
```

```
c0105000 t rest_init
c0105020 t do_pre_smp_initcalls
c0105030 t run_init_process
c0105060 t init
. . . .
```

```
e0c197e0 t journal_write_revoke_records [jbd]
96d91461 a __crc_journal_dirty_data    [jbd]
c0129650 U __mod_timer [jbd]
c0141e30 U __kmalloc [jbd]
c01415c0 U kmem_cache_destroy [jbd]
c0154d60 U bh_waitq_
head [jbd]
[rob@localhost rob]
```

```

/* kello.c kernel module,
- gcc -c -DMODULE kello.c
- switch to root, using su
- insert module using /sbin/insmod -f ./kello.o
- remove using /sbin/rmmod
- exit from root
*/
#define MODULE
#define __KERNEL__
#include <linux/module.h>
#include <linux/init.h>

module_init(testit_init);
module_exit(testit_exit);

int testit_init(void) {
    printk("<1>Hello from kworld!!\n");
    return 0;
}

void testit_exit(void) {
    printk("<1>Goodbye kworld??\n");
}

```

To compile and run the kernel module use the following instructions:

```

[rob@localhost] su
password *****
[root@localhost] gcc -c -O2 -Wall -DMODULE hello.c
[root@localhost] insmod -f ./hello.o
    Hello from kworld!!
[root@localhost] rmmod hello
    Goodbye kworld??
[root@localhost]

```

```

struct resource {
    const char *name;
    unsigned long start, end;
    unsigned long flags;
    struct resource *parent, *sibling, *child;
};

struct resource *request_region(unsigned long base_port,
                                unsigned long port_range,
                                char *name);
void release_region(unsigned long base_port,
                    unsigned long port_range);

```

```

struct file_operations {
    struct module *owner;
    loff_t      (*llseek) (struct file*, loff_t, int);
    ssize_t     (*read) (struct file*, char __user*, size_t, loff_t*);
    ssize_t     (*aio_read) (struct kiocb*, char __user*, size_t, loff_t);
    ssize_t     (*write) (struct file*, const char __user*, size_t, loff_t*);
    ssize_t     (*aio_write) (struct kiocb*, const char __user*, size_t, loff_t);
    int         (*readdir) (struct file*, void*, filldir_t);
    unsigned int (*poll) (struct file*, struct poll_table_struct*);
    int         (*ioctl) (struct inode*, struct file*, unsigned int, unsigned long);
    int         (*mmap) (struct file*, struct vm_area_struct*);
    int         (*open) (struct inode*, struct file*);
    int         (*flush) (struct file*);
    int         (*release) (struct inode*, struct file*);
    int         (*fsync) (struct file*, struct dentry*, int datasync);
    int         (*aio_fsync) (struct kiocb *, int datasync);
    int         (*fasync) (int, struct file*, int);
    int         (*lock) (struct file*, int, struct file_lock*);
    ssize_t     (*readv) (struct file , const struct iovec*, unsigned long, loff_t*);
    ssize_t     (*writev) (struct file*, const struct iovec*, unsigned long, loff_t*);
    ssize_t     (*sendfile) (struct file*, loff_t*, size_t, read_actor_t, void __user*);
    ssize_t     (*sendpage) (struct file*, struct page*, int, size_t, loff_t*, int);
    unsigned long(*get_unmapped_area)(struct file*, unsigned long,
                                      unsigned long, unsigned long, unsigned long);
};

```

```

struct file_operations my_fops = {
    llseek: my_llseek,
    read: my_read,
    write: my_write,
    ioctl: my_ioctl,
    open: my_open,
    release: my_release,
};

. . .
if (nret = register_chrdev(MY_MAJOR_NUM, "mydev", &my_fops)) < 0)
    printk(KERN_ERR "register_chrdev: %d\n", nret);

struct file {
    struct list_head f_list;
    struct dentry *f_dentry;           //directory entry, gives access to inode
    struct vfsmount *f_vfsmnt;
    struct file_operations *f_op; //pointer to device specific operations
    atomic_t f_count;
    unsigned int f_flags;               //file flags from open()
    mode_t f_mode;                     //read or write access
    loff_t f_pos;                      //offset of pointer into data
    struct fown_struct f_owner;
    unsigned int f_uid, f_gid;
    int f_error;
    struct file_ra_state f_ra;
    unsigned long f_version;
    void *f_security;
    void *private_data; //needed for tty driver
    struct list_head f_ep_links;
    spinlock_t f_ep_lock;
    void *f_supermount;                //used by supermount
};

new_maj_num = register_chrdev(0, //request a new major number
    const char *dev_name,
    struct file_operations * my_fops);

```

```

#!/bin/sh
# based on chapt 3 example, Rubini & Corbet

module="my_module"
device="my_device"
mode="644"
group="staff"

# insert module and accept a major number
/sbin/insmod -f ./ $module.o || exit 1

rm -f /dev/${device}[0-3]          #delete stale drivers

# find and record the new major number
major_num = `gawk '/'$module'/ {print $1}' /proc/devices`

# create special device file inodes
mknod /dev/${device}0 c $major 0
mknod /dev/${device}1 c $major 1
mknod /dev/${device}2 c $major 2
mknod /dev/${device}3 c $major 3

# set file permissions
chgrp $group /dev/${device}[0-3]
chmod $mode /dev/${device}[0-3]

unsigned long __copy_to_user_ll(void __user *to,
                                const void *from,
                                unsigned long n);

static inline unsigned long __copy_to_user(void __user *to,
                                           const void *from,
                                           unsigned long n) {
    if (__builtin_constant_p(n)) {
        unsigned long ret;
        switch (n) {
            case 1:
                __put_user_size(*(u8 *)from, (u8 *)to, 1, ret, 1);
                return ret;
            case 2:
                __put_user_size(*(u16 *)from, (u16 *)to, 2, ret, 2);
                return ret;
            case 4:
                __put_user_size(*(u32 *)from, (u32 *)to, 4, ret, 4);
                return ret;
        }
    }
    return __copy_to_user_ll(to, from, n);
}

static inline unsigned long copy_to_user(void __user *to,
                                          const void *from,
                                          unsigned long n) {
    might_sleep();
    if (access_ok(VERIFY_WRITE, to, n))
        n = __copy_to_user(to, from, n);
    return n;
}

unsigned long __copy_from_user_ll(void *to,
                                  const void __user *from,
                                  unsigned long n);

static inline unsigned long

```

```
if (__builtin_constant_p(n)) {
    unsigned long ret;

    switch (n) {
    case 1:
        __get_user_size(*(u8 *)to, from, 1, ret, 1);
        return ret;
    case 2:
        __get_user_size(*(u16 *)to, from, 2, ret, 2);
        return ret;
    case 4:
        __get_user_size(*(u32 *)to, from, 4, ret, 4);
        return ret;
    }
}
return __copy_from_user_ll(to, from, n);
}

static inline unsigned long
copy_from_user(void *to, const void __user *from, unsigned long n) {
    might_sleep();
    if (access_ok(VERIFY_READ, from, n))
        n = __copy_from_user(to, from, n);
    else
        memset(to, 0, n);
    return n;
}
```

cdroms	console	discs	dsp	floppy	full
ide	initctl	input	kmem	kmsg	log
mem	misc	mouse	null	port	psaux
prntrs	ptmx	pts	pty	random	rd
root	rtc	scsi	seqnrc	shm	snd
sound	random	usb	vc	vcc	vcs

Fig. 19.8 The new Linux devfs

20. Hardware/Software Co-design

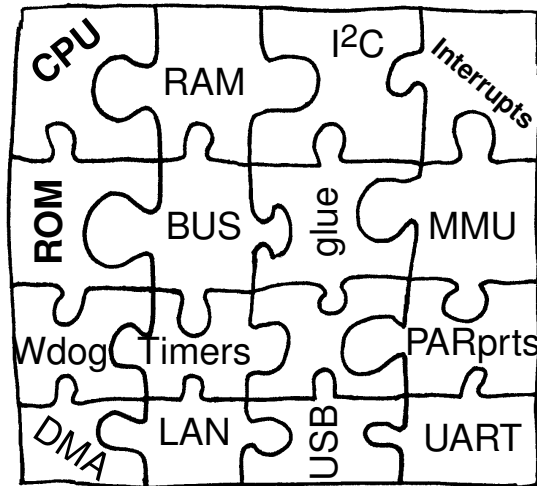


Fig. 20.2 Solving the IP jigsaw puzzle

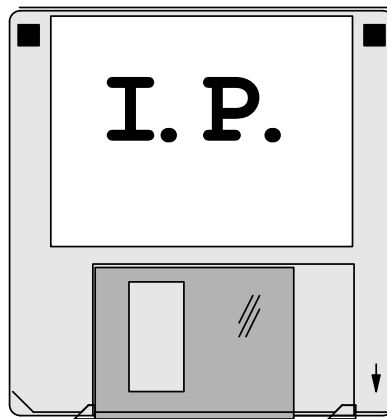
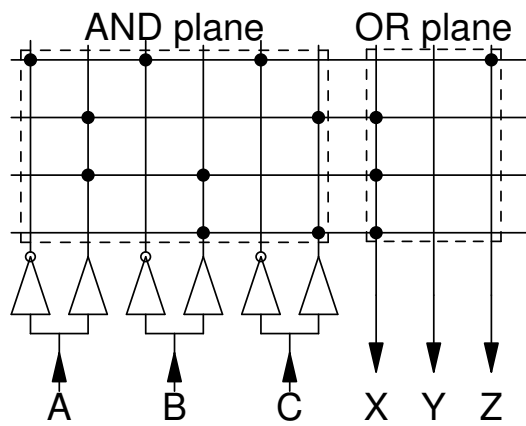


Fig. 20.3 Si.ware



$$X = B \cdot C + A \cdot B + A \cdot C$$

$$Z = \bar{A} \cdot \bar{B} \cdot \bar{C}$$

Y – unused

Fig. 20.4a Three term AND-OR product term matrix

- 10^9 transistors
- 4×10^7 logic gates
- packages with 2000 pins
- 1.5 Gbps I/O data rates
- upto 1 GHz CPU clock
- 1.0 V core voltage
- 24 MB internal RAM
- 16×10^7 configuration bits

Fig. 20.4b Future FPGA capacity and performance

- Bespoke microcontrollers, using a standard CPU core, but customised with specialized peripheral interfaces.
- SMP, symmetric multi-processor compute engines for dedicated algorithmic solutions.
- Multi-tasking platforms, offering one CPU per task: true parallel processing.
- High performance synchronous logic, migrating an existing application from sequential software executed on a processor, to dedicated hardware.
- Secure application requirements, where conventional ROM software would be too vulnerable to theft. The FPGA configuration file is encrypted and stored in ROM for loaded into the FPGA at power-on, using a OTP decryption microcontroller.
- Dynamically self-reconfigurable application hardware, such as switching between encryption and decryption, or compression and decompression, functionality.
- Application specific instruction set extensions, adding extra functionality to an existing 'soft' CPU. In the same manner that CISC microcode could be upgraded to include new instructions.

Fig. 20.4c Future applications for large FPGAs

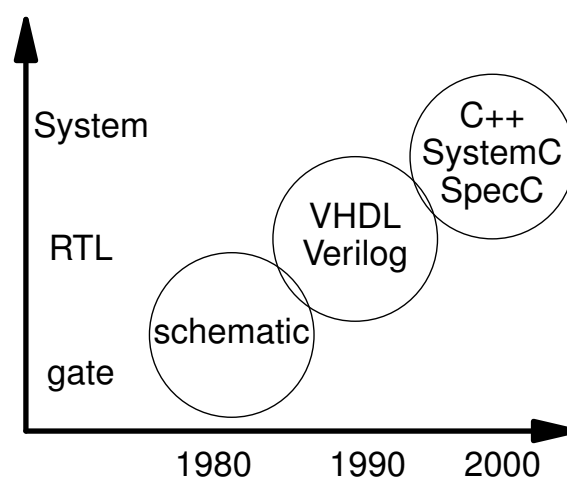


Fig. 20.5 Trends in chip modelling

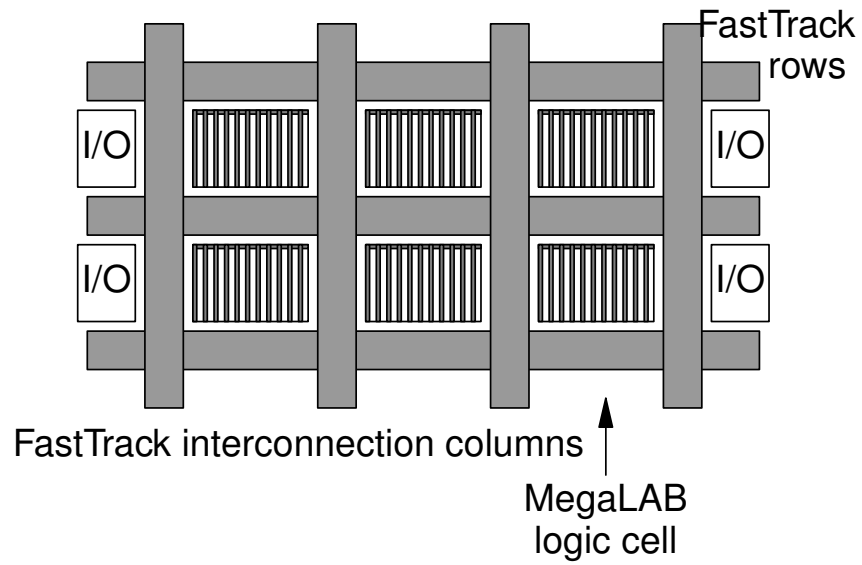


Fig. 20.6a Block layout of an Altera APEX20K cPLD

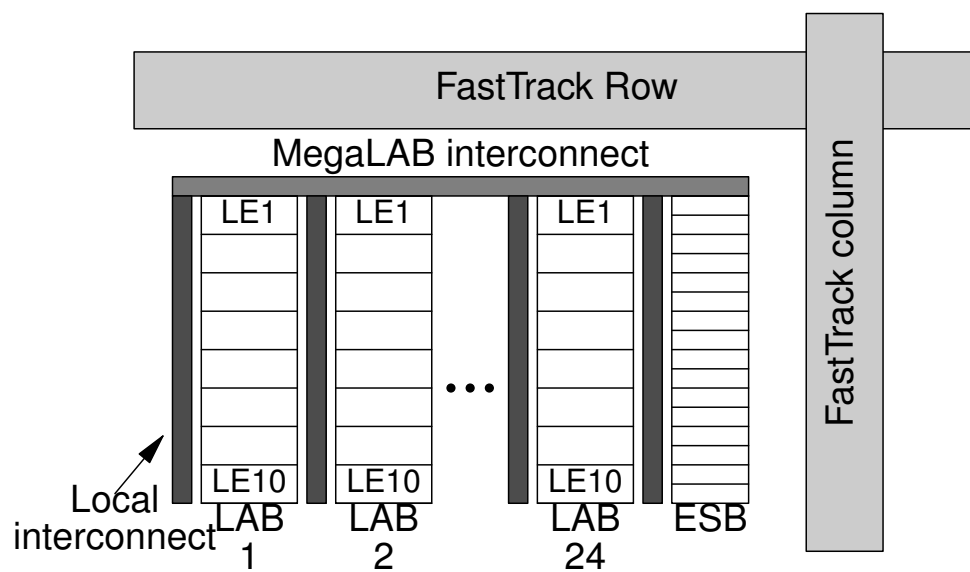


Fig. 20.6b A single MegaLAB logic cell from an Altera APEX20K cPLD

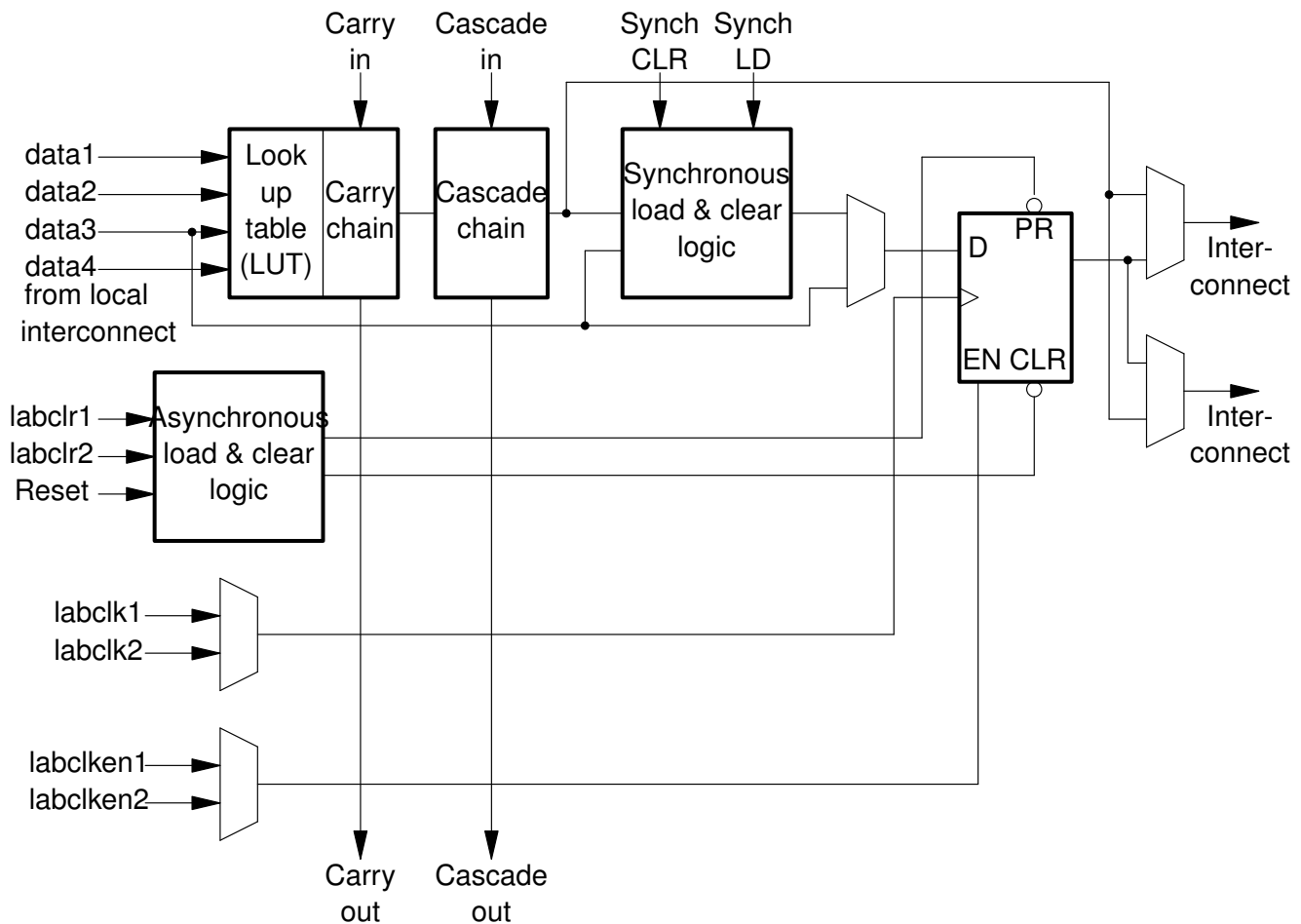


Fig. 20.6c APEX20k Logic Element (LE)

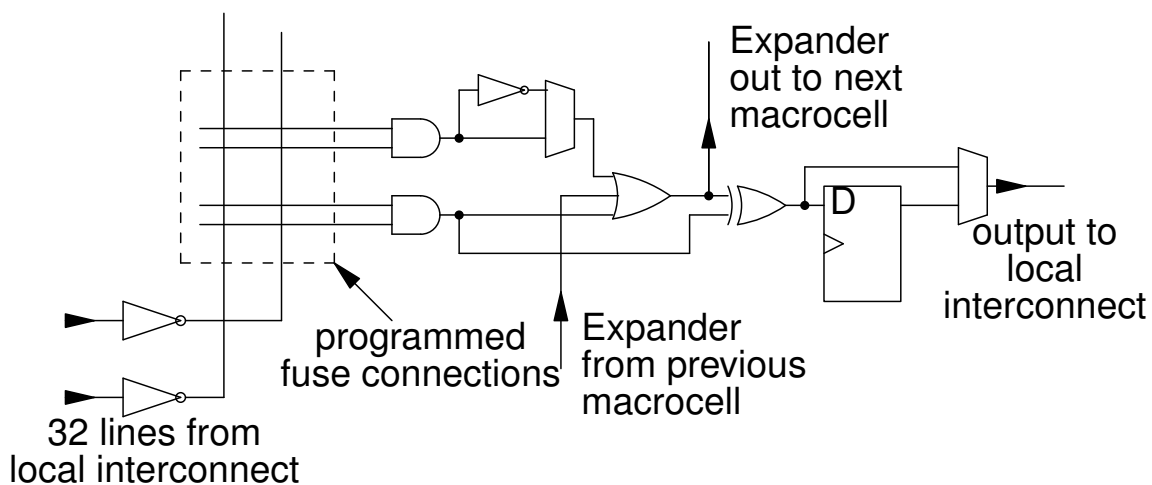


Fig. 20.6d APEX20k Product term Macrocell