EE244: Design Technology for Integrated Circuits and Systems

Outline
Lecture 2.1

◆ Representing the Design: Hierarchy, Abstraction
◆ Netlists
◆ The EDIF Design Interchange format.

Design Representation

◆ Some variation of entity-relationship model most common today.
◆ Data model usually based on existing interchange format or design language (e.g. EDIF, VHDL).
◆ Open Issue: Forced Consistency (conventional database approach) versus Periodic Check (C "Lint" approach).
◆ Integration Environment provides lingua franca for design representation. Useful side-effect of standards meetings is a common understanding of terms and data objects.
Abstracting Layouts

Terminal Frame

Port D

Pin

Protected on all Mask layers

Protection Frame
Abstracting Layouts

The Netlist View

- List of parts used in a design and their electrical connections:
- net-oriented (e.g. EDIF)
  - partName, partType, portNames (perhaps implicit)
  - netName, ports (or pins) it connects
    - (instance Dlatch latch1)
    - (instance Dlatch latch2)
    - (net (joined (portReference latch1 D) (portReference latch2 Q)))
- or part-oriented (e.g. Spice)
  - partName, nets connected to, partType
    - M1 1 2 3 11 NMOS
    - M2 4 5 2 11 PMOS
Entity-Relationship Approach to Data Representation

- cellInstance
- portInstanceOf
- net
- connects
- portOf
- instanceOf
- cell
- name
- capacitance
- port
- name
- capacitance

What's in a Name?

- Just about Everything!
- Efficient name resolution - resolving references to design objects - is one of the most important, "undecided" research problems.
- Strongly related to multiprocessor distributed cache consistency problem, distributed file system problem, general distributed data management problem.
- Ultimate issue is efficient pruning of "global search." (replication of read-only data, use of "hints," management of domains and dynamic data migration are all important.)
Introduction to EDIF

(int def cellNameDef a named cell definition
  < viewMap > | relates ports & instances of views
  { view | different views of the cell
    property } properties of the cell
}

Introduction to EDIF: Cells and Views

(cell DLatch
  (view NETLIST NetList
    (interface
      abstraction of the cell
    )
    (contents
      implementation of the cell
    )))
Introduction to EDIF: Abstraction

(interface
  (body PROTECTION_FRAME
    (portImplementation D (figureGroup …)
    (portImplementation E (figureGroup …)
    (portImplementation Q (figureGroup …)
    (figureGroup GREEN (rectangle (point x y) ..))
  )
  (contents
    lots of figure groups containing rectangles
  )
))

Introduction to EDIF: Contents

(cell TwoLatches
  (view NETLIST NetList
    (interface abstraction of the cell )
    (contents
      (instance DLatch NetList Latch1)
      (instance DLatch NetList Latch2)
      (net Net1 (joined
        (portReference Latch1 D)
        (portReference Latch2 Q))
  )))

Introduction to EDIF: Netlists

(contents
  (instance DLatch NetList Latch1)
  (instance DLatch NetList Latch2)
  (net Net1 (joined
    (portReference Latch1 D)
    (portReference Latch2 Q)))))
)
instance: a1

master: dataPath

(EDIF:portInstance)

(actual pin)

formal terminal: "VDD!"

(EDIF: port)

master: ALU

formal pin

formal terminal: "VDD!"

(EDIF: (cell (view (contents))))

(OCT: cell: view: contents)

(EDIF: (cell (view (contents)))

(master: dataPath)

(actual terminal: "VDD!"

(EDIF:portInstance))

(formal terminal of datapath)

- newton
- chips
- nmos1
- scmos
- fast
- slow
- posBrain
- neuron
- leg
- dataPath
- ~cad/lib/technology

Copyright © 1997, A. Richard Newton
chips
nmos1 scmos
fast slow
posBrain
dataPath
neuron
leg
dataPath
ALU
ALU
symbolic
symbolic
symbolic
current
current
contents
contents
contents
contents
contents

chip
dataPath
REG
n1
nand
a1
ALU
n1
nand

attribute of port instance "output"
Introduction to EDIF: Nested Nets

(instance DLatch Symbolic Latch1)
(instance DLatch Symbolic Latch2)
(net Net1 (joined
  (portReference Latch1 D)
  (portReference Latch2 Q))
(instance M1M2 Symbolic c1)
(net netseg1 (joined
  (portReference Latch1 D)
  (portReference C1 viaPort))
(net netseg1 (joined
  (portReference Latch2 Q)
  (portReference C1 viaPort))))

Managing Access to Data

Example: The OCT Approach
OctObject Structure

```c
struct octObject {
    octObjectType type;
    octId objectId;
    union {
        struct octFacet facet;
        struct octInstance instance;
        struct octProp prop;
        struct octTerm term;
        struct octNet net;
        struct octBox box;
        struct octPolygon polygon;
        struct octCircle circle;
        struct octPath path;
        struct octLabel label;
        struct octBag bag;
        struct octLayer layer;
        struct octPoint point;
        struct octEdge Edge;
        struct octChangeList changeList;
        struct octChangeRecord changeRecord;
    } contents;
};
```

The OctFacet Object

```c
struct octFacet {
    /* facet object */
    char *cell;     /* cellName */
    char *view;     /* viewName */
    char *facet;    /* "interface" or "contents" */
    char *version;  /* OCT_CURRENT_FACET */
    char *mode;     /* "r", "w" or "a" */
};
```
The octPoint and octBox Objects

```c
struct octPoint {
    /* oct Point */
    octCoord x; /* x coordinate (32-bit int) */
    octCoord y; /* y coordinate (32-bit int) */
};

struct octBox { /* oct Box */
    struct octPoint lowerLeft;
    struct octPoint upperRight;
};
```

Operations on Facets

```c
void octBegin()
void octEnd()

octStatus octOpenFacet(octObject *facet)
octStatus octCloseFacet(octObject *facet)

octStatus octOpenMaster(octObject *instance, *facet)
octStatus octOpenRelative(octObject *rfacet, *facet, int location)
octStatus octFlushFacet(octObject *facet)
octStatus octWriteFacet(octObject *new, *old)
octStatus octCopyFacet(octObject *new, *old)
octStatus octFreeFacet(octObject *facet)
octStatus octGetFacetInfo(octObject *facet, struct octFacetInfo *info)
octFullName(octObject *facet, char **name)
```
Operations on Data Items

octStatus octCreate(octObject *cnt, *obj)
octStatus octDelete(octObject *obj)
octStatus octModify(octObject *obj)

octStatus octAttach(octObject *cnt, *obj)
octStatus octDetach(octObject *cnt, *obj)

octStatus octAttachOnce(octObject *cnt, *obj)
octStatus octIsAttached(octObject *cnt, *obj)

octStatus octPutPoints(octObject *obj, int32 num, octPoint *pnts)
octStatus octGetPoints(octObject *obj, int32 *num, octPoint *pnts)

Retrieving Data Items

octStatus octInitGenContents(
    octObject *cnt, octObjectMask mask, octGenerator *gen)
octStatus octInitGenContainers(
    octObject *obj, octObjectMask mask, octGenerator *gen)
octStatus octGenerate(octGenerator *gen, octObject *obj)

Values for mask:

<table>
<thead>
<tr>
<th>OCT_FACET_MASK</th>
<th>OCT TERM MASK</th>
</tr>
</thead>
<tbody>
<tr>
<td>OCT_NET_MASK</td>
<td>OCT INSTANCE MASK</td>
</tr>
<tr>
<td>OCT_PROP_MASK</td>
<td>OCT BAG MASK</td>
</tr>
<tr>
<td>OCT_POLYGON_MASK</td>
<td>OCT BOX MASK</td>
</tr>
<tr>
<td>OCT_CIRCLE_MASK</td>
<td>OCT PATH MASK</td>
</tr>
<tr>
<td>OCT_LAYER_MASK</td>
<td>OCT LAYER MASK</td>
</tr>
<tr>
<td>OCT_POINT_MASK</td>
<td>OCT EDGE MASK</td>
</tr>
<tr>
<td>OCT_FORMAL MASK</td>
<td>OCT CHANGE LIST MASK</td>
</tr>
<tr>
<td>OCT_CHANGE_MASK</td>
<td></td>
</tr>
<tr>
<td>OCT_CHANGE_RECORD_MASK</td>
<td></td>
</tr>
</tbody>
</table>
Use of Generators

/* proper way to generate */
while (octGenerate(&gen, &obj) == OCT_OK) {
    /* do something */
}

/* XXX wrong way to generate */
while (octGenerate(&gen, &obj) != OCT_GEN_DONE) {
    /* do something */
}

Generator Examples

octInitGenContents(&facet, OCT_NET_MASK, &gen);
while (octGenerate(&gen, &net) == OCT_OK) {
    /* do something */
}

/* XXX will loop infinitely */
newnet.type = OCT_NET;
newnet.contents.net.name = "new net";
octInitGenContents(&facet, OCT_NET_MASK, &gen);
while (octGenerate(&gen, &net) == OCT_OK) {
    octCreate(&facet, &newnet);
}
OCT Operations and the Environment

```
#include "copyright.h"
#include "port.h"
#include "oct.h"

main(argc, argv)
int argc;
char **argv;
{
    /* declare the oct objects to be used */
    octObject facet; /* facet to be opened */
    octObject layer; /* layer containing the geometry */
    octObject geo; /* geometry on the layer */

    /* declare the oct generators to be used */
    octGenerator lgen; /* generator for the layers */
    octGenerator ggen; /* generator for the geometries */

    /* initialize oct - allocate tables, notify design managers, etc. */
    octBegin();
}```
/*
 * open the facet
 */
facet.type = OCT_FACET;
facet.contents.facet.cell = argv[1];
facet.contents.facet.view = argv[2];
facet.contents.facet.facet = "contents";
facet.contents.facet.version =
OCT_CURRENT_VERSION;
facet.contents.facet.mode = "r";
if (octOpenFacet(&facet) < OCT_OK) {
    octError("opening facet to be generated");
    exit(-1);
}

/* generate over all layers */
(void) octInitGenContents(&facet, OCT_LAYER_MASK, &lgen);
while (octGenerate(&igen, &layer) == OCT_OK) {
    /*
     * generate over all geometries on the layer
     */
    (void) octInitGenContents(&layer, OCT_GEO_MASK, &ggen);
    while (octGenerate(&ggen, &geo) == OCT_OK) {
        /*
         * process the geometry
         */
    }
} /* close down oct - release memory, notify design managers, etc. */
octEnd();
exit(0);
/*
generate over all layers
*/

(void) octInitGenContents(&facet, OCT_LAYER_MASK, &lgen);
while (octGenerate(&igen, &layer) == OCT_OK) {

    /*
    * generate over all geometries on the layer
    */

    (void) octInitGenContents(&layer, OCT_GEO_MASK, &ggen);
    while (octGenerate(&ggen, &geo) == OCT_OK) {

        /*
        * process the geometry
        */
    }
}