MANCHESTER 1824

The University of Manchester

Balsa

Behaviour to Silicon

Doug Edwards

School of Computer Science
University of Manchester

MANCHESTER 1824

The University of Manchester

Overview of Presentations

- Demonstration of Balsa system (Doug Edwards)
  - quick overview
  - building implementations
- Current Balsa Optimisations (Luis Tarazona)
  - description strategies
  - handshake circuit optimisations
- New Teak synthesis system (Andrew Bardsley)
  - creating new back-ends using ABS

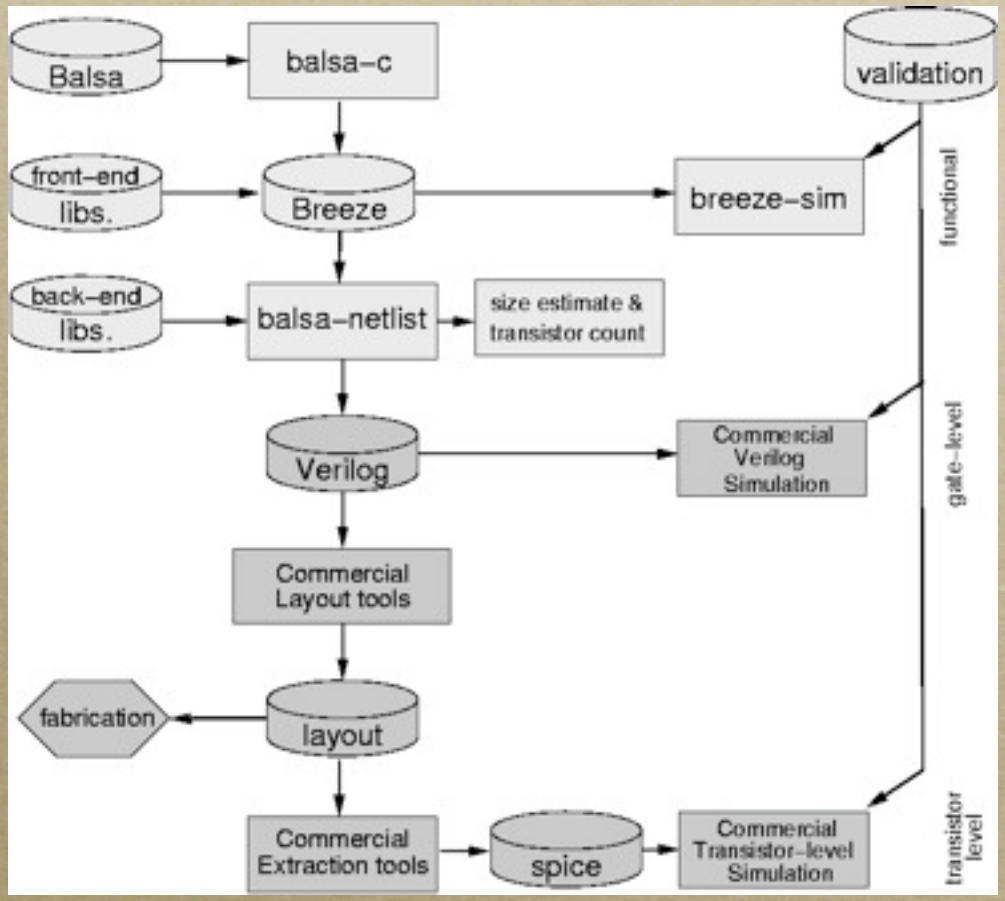
MANCHESTER 1824

The University of Manchester

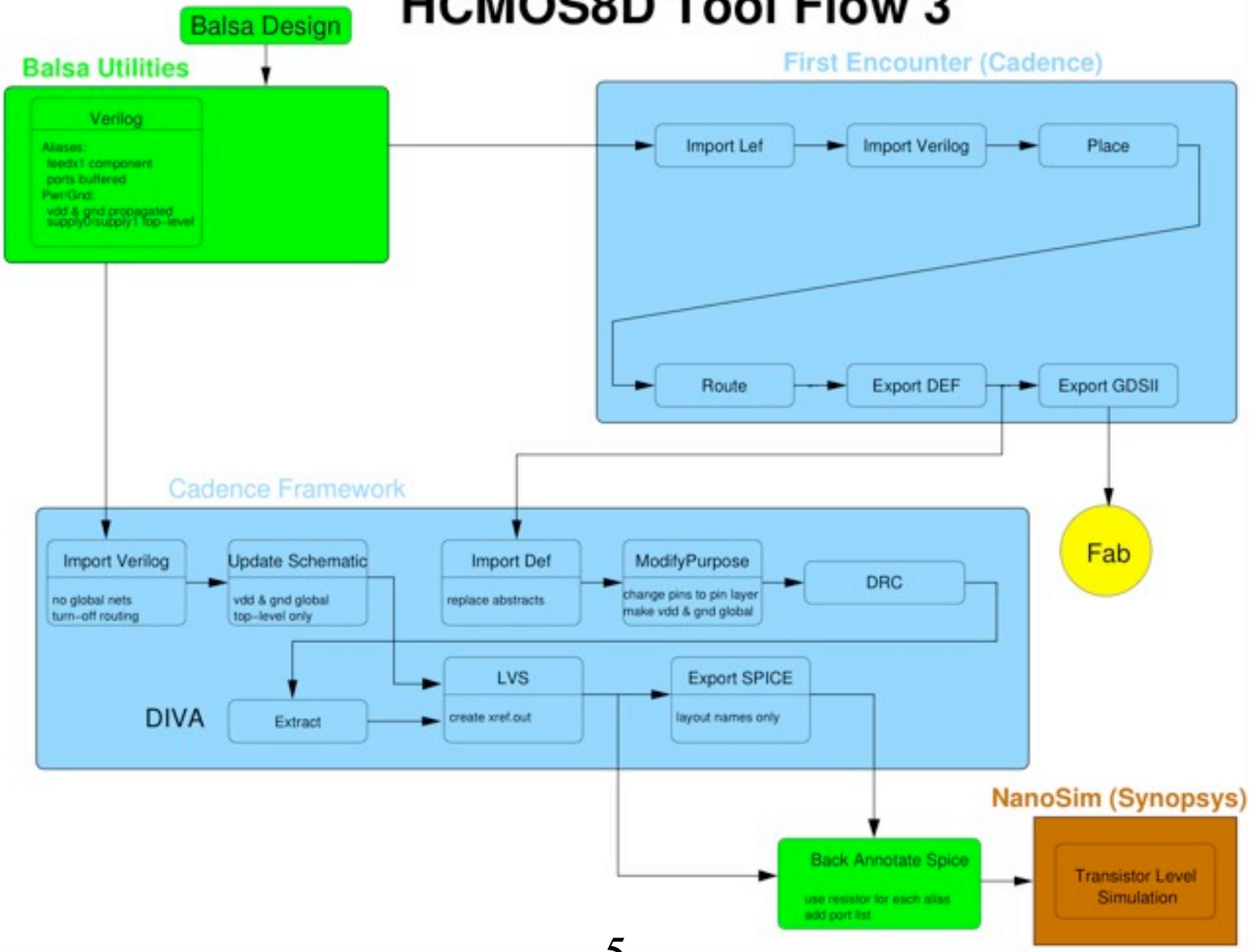
Balsa

- Synthesis framework for self-timed circuits
- Compiles a behavioural description to a set of (~40) intermediate *handshake components*
- Syntax Directed Compilation
  - Language constructs map to handshake components
- Different implementation of handshake components for different protocols and technologies
- Builtin functions for constructing test suites in Balsa

Balsa Design Flow



HCMOS8D Tool Flow 3



MANCHESTER 1824

The University of Manchester

Balsa Back-Ends

- 180 nm ST, generic Verilog and Xilinx libraries,
- Single rail, dual rail & 1-of-4 in standard distribution
  - Arbitrary n-of-m codes can also be generated
- Handshake components described by abstract gate operators.
- Complex cells, such as adders, can be described from cells in the library.
- The netlist is then mapped according to specifications defined by the technology
- New technologies may be created using a Balsa utility: *balsa-make-helpers* which generates any cells not resident in library.

MANCHESTER 1824

The University of Manchester

Next (Public) Release

- Optimised handshake components
- Buffer Insertion
- GUI elements moved to GTK2
- Style options reorganised
- Scheme directory reorganised
- balsa-netlist rewritten
- Obsolete parts of release removed

MANCHESTER 1824

The University of Manchester

Progress against SPA

- x13 improvement
  - simplified architecture (e.g. no Thumb)
  - fixing bugs in description (e.g. 7-port register bank)
  - “bottom up” description - pipeline style
  - new handshake components
  - architecture improvements (e.g. forwarding)

MANCHESTER 1824

The University of Manchester

Balsa Walkthrough

- Simple Buffer example
  - balsa-mgr
  - balsa utilities
  - adding a test harness & balsa-level simulations
- SSEM example
  - balsa builtin functions
  - choosing different implementation styles
  - verilog simulation
  - interfacing to Cadence - generating a layout

MANCHESTER 1824

The University of Manchester

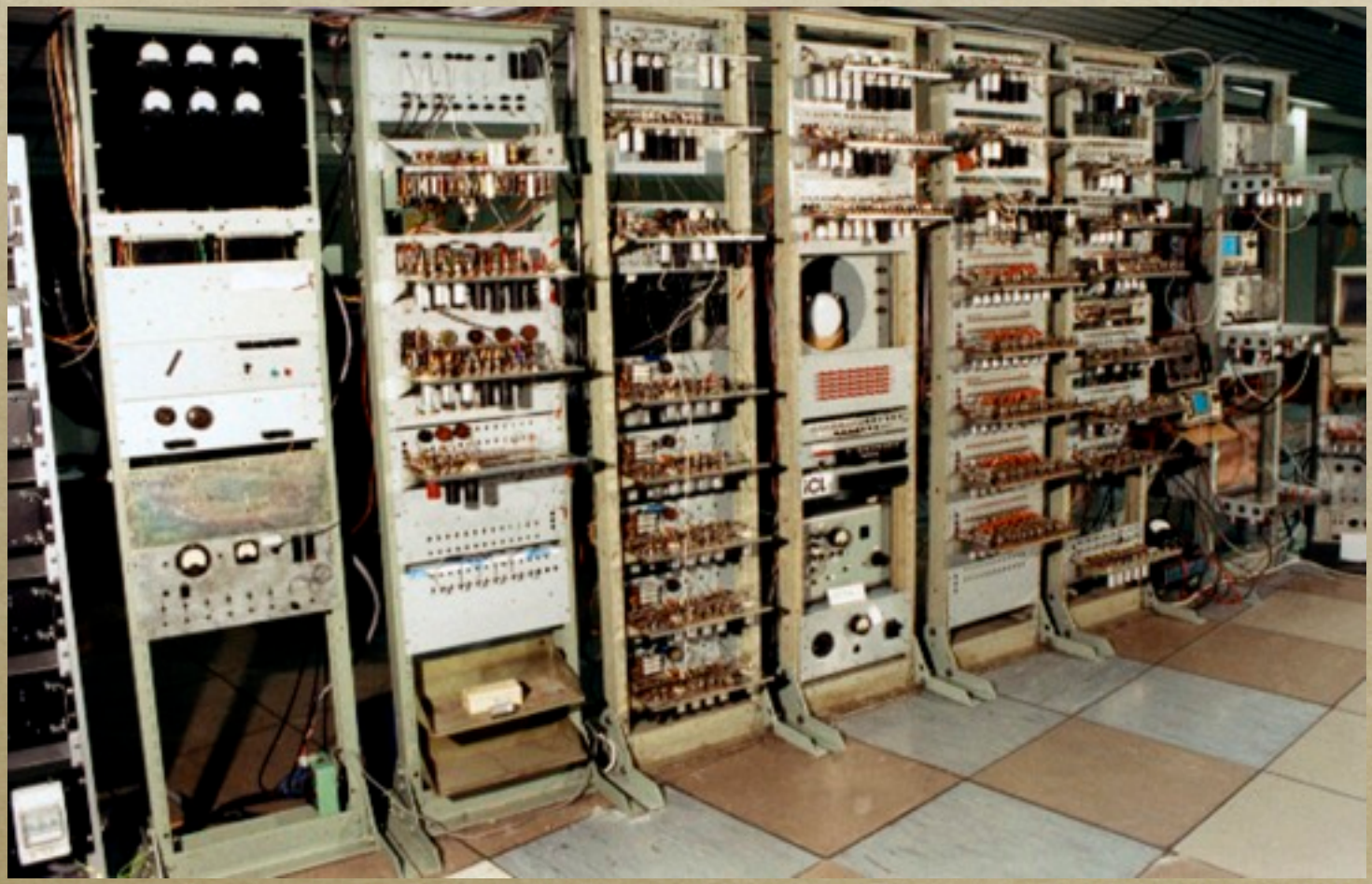
SSEM - the "Baby"

- SSEM (Small Scale Experimental Machine or "Baby")
  - World's 1st stored program machine
  - ran GCD program 21st June 1948 in Manchester
  - 32 bit processor, 2's complement, 32 word memory
  - 7 instruction types
  - Single register accumulator (ACC)
  - program counter (PC)

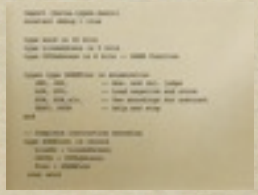
MANCHESTER 1824

The University of Manchester

The Baby (rebuild)



Balsa Baby Demo



- Peek at Balsa description
- Compile the description
- Run Balsa test bench
  - source code in *gcd.s*, precompiled to file *prog*
  - program computes the GCD of locations 0x11 and 0x12
    - defined as 0xC and 0x8
  - Result returned in location 0x11 (= 4)

MANCHESTER 1824

The University of Manchester

Layout Demonstration

- Dual-rail implementation in 180nm technology
  - verilog netlist produced
  - transistor level (nanosim) simulation
- Place and Route using Cadence Encounter
- Repeat for single Nor gate primitive library
  - in same 180 nm technology

Simulation Demonstration

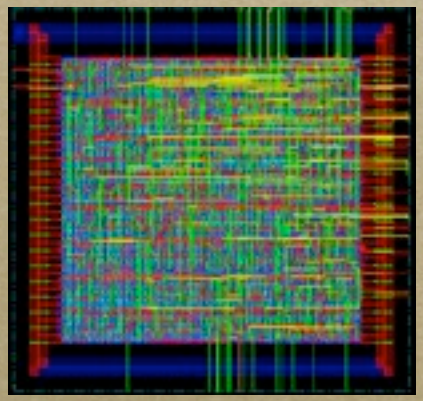
- Balsa generated verilog of SSEM used
  - but uses a custom verilog test-bench
    - porting of Balsa test-bench not complete

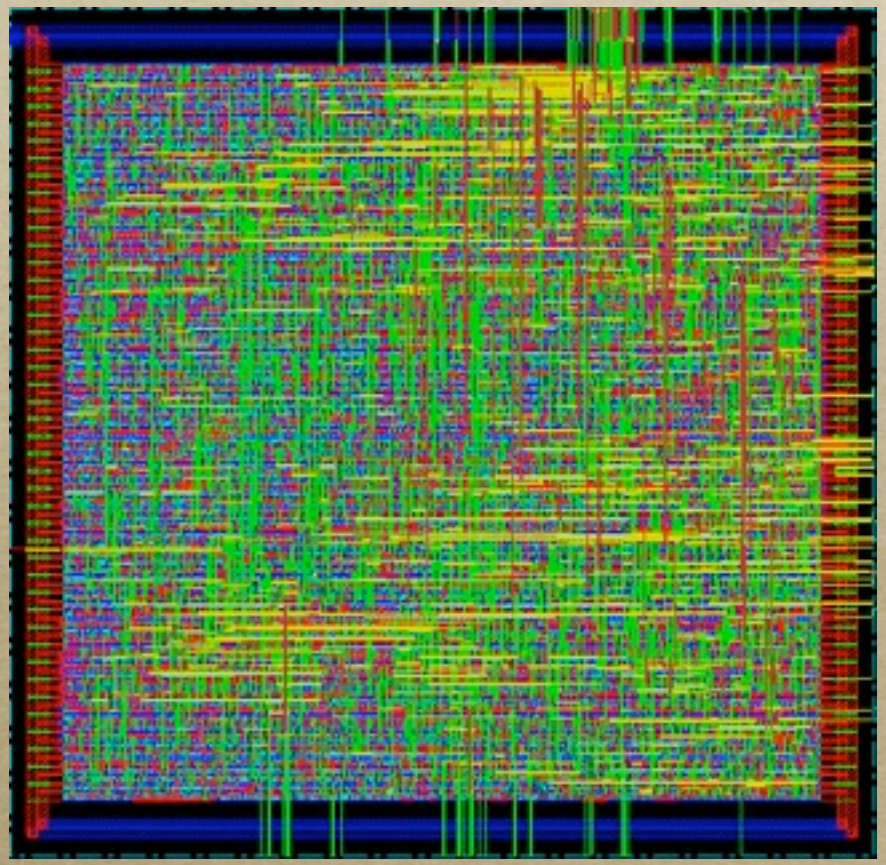
1. Verilog (unit gate delay) simulation
2. Nanosim transistor-level simulation of the two implementations
   - spice file of layout generated off-line
     - DIVA extractor is obsolete, doesn't run on latest linux kernels

MANCHESTER 1824

The University of Manchester

Comparative Layouts





MANCHESTER 1824

The University of Manchester

Acknowledgements

- Andrew Bardsley: Balsa System
- Will Toms: back-end(s) and demo assistance
- Lililan Janin: balsa-mgr and simulation tool
- Luis Plana: Optimisations
- Luis Tarazona: Buffer Insertion & Optimisations
- Jeff Pepper: Cadence/Synopsys guru and general tool support.
- School of CS & EPSRC for Balsa funding

```
import [balsa.types.basic]
constant debug = true

type word is 32 bits
type LineAddress is 5 bits
type CRTAddress is 8 bits -- SSEM function

types type SSEMFunc is enumeration
   JMP, JRP,          -- Abs. and rel. jumps
   LDN, STO,          -- Load negative and store
   SUB, SUB_alt,      -- Two encodings for subtract
   TEST, STOP         -- Skip and stop
end

-- Complete instruction encoding
type SSEMInst is record
   LineNo : LineAddress;
   CRTNo : CRTAddress;
   Func : SSEMFunc
 over word
```

```
-- SSEM: Top level
procedure SSEM (
  -- Memory interface, MemA,MemRNW,MemR,MemW
  output MemA : LineAddress;
  output MemRNW : bit;
  input MemR : word;
  output MemW : word ;
  -- Signal halt state
  sync halted
) is

  variable ACC, ACC_slave : word
  variable IR : word
  variable PC, PC_step : LineAddress
  variable MDR : word
  variable Stopped : bit
```

```
-- Extract an address from a word
  function ExtractAddress (wordVal : word) =
  (wordVal as SSEMInst).LineNo

  shared WriteExtractedAddress is begin
    MemA <- ExtractAddress (IR) end

  -- Memory operations, shared procedures
  shared MemoryWrite is
  begin MemRNW <- 0 || WriteExtractedAddress ()
 || MemW <- ACC_slave end

  shared MemoryRead is
  begin MemRNW <- 1 || WriteExtractedAddress ()
 || MemR -> MDR end

  -- Fetch an instruction IR := M[PC]
  procedure InstructionFetch is
  begin MemRNW <- 1 || MemA <- PC || MemR -> IR end
```

```
-- Modify the programme counter PC
  shared IncrementPC is begin
    PC := (PC + PC_step as LineAddress) end
  shared AddMDRToPC is begin
    PC_step := ExtractAddress (MDR); IncrementPC () end

 procedure DecodeAndExecuteInstruction is
  begin
 case (IR as SSEMInst).Func of
      JMP then MemoryRead (); ZeroPC (); AddMDRToPC ()
    | JRP then MemoryRead (); AddMDRToPC ()
    | LDN then ZeroACC (); SUB ()
    | STO then MemoryWrite ()
    | SUB .. SUB_alt then SUB ()
    | TEST then
      if #ACC [31] -- -ve?
      then IncrementPC () end
    | STOP then Stopped := 1
    end ;
    ACC := ACC_slave
  end
```

```
begin
  ZeroACC () || ZeroPC () ||
  Stopped := 0; -- reset initialisation
  loop while not Stopped then
    PC_step := 1;
    IncrementPC ();
    InstructionFetch ();
    DecodeAndExecuteInstruction ()
  end ; -- loop
  sync halted
  -- halt -- STOP instruction effect
end
```

back to demo