Describing Handshake Components with ABS

Andrew Bardsley
School of Computer Science
The University of Manchester

Overview

- The (planned) structure of the new Balsa release
- The ABS language for describing HCs
- Build a new style package: balsa-style-sync with synchronous HCs
- Other ways into balsa-netlist

Balsa tools/packages

- Who needs to know about HC/protocols:
  - balsa-c - components are hard-coded
  - balsa-netlist - needs ABS descriptions to make HCs, port structures to compose ‘part’ netlists
  - balsa-make-impl-test - needs to know about protocols

Directory structure

- /bin/{balsa-c, balsa-netlist ...}
- /share
  - style/{four_b_rb, dual_b, sync}
  - config/{balsa, balsa-style-sync}
  - tech/{example, xilinx ...}
  - scheme/{balsa-netlist.scm, ...}
- Packages add files to installation

Style packages

- balsatech:TECH/STYLE/STYLE-OPTS
- Style describes signalling protocol, component choice
- Builtin styles include: four_b_rb, dual_b
- Style-options give further refinement
- I'll be using: example/sync

balsa-style-sync

- A separate package describing out new components. Configured using automake
- Only need four files:
  - Makefile.am, configure.ac - for autotools
  - balsa-style-sync/startup.scm
  - balsa-style-sync/balsa-style-sync

Makefile.am

- balsaconfigdir = $(datadir)/config
  stylesyncdir = $(datadir)/style/sync

```
balsaconfig_DATA = balsa-style-sync
stylesync_DATA = startup.scm

EXTRA_DIST = balsa-style-sync \
    startup.scm
```

configure.ac

- AC_INIT(sync/startup.scm)
  AM_INIT_AUTOMAKE(balsa-style-sync, 0.0)
  AC_PROG_INSTALL
  AC_PATH_PROG(BALSA_CONFIG, balsa-config)

```
if test ${BALSA_CONFIG}; then : ; else
   AC_MSG_ERROR([cannot find balsa-config]); fi
AC_MSG_CHECKING(installed Balsa directory)
ac_default_prefix=`${BALSA_CONFIG} -d`
prefix=${ac_default_prefix}
AC_MSG_RESULT(${ac_default_prefix})
AC_OUTPUT([Makefile])
```

balsa-style-sync

- Package description for balsa-config
- No fixed format. Just a textual description
- Existence of share/config/STYLE used to check for package installation
- Contents: “balsa-style-sync: prototype synchronous back-end”

startup.scm

- Scheme file run after reading components.abs
- Can change:
  - channel composition
  - choice of components
  - do arbitrary other things
- Most functions taken from: brz-tech

startup.scm: channels

- (set! tech-sync-channel-portions
  `((req push ,width-one)
  (ack pull ,width one)))
- (set! tech-push-channel-portions
  `((req push ,width-one)
  (ack pull ,width-one)
  (data push ,width-n)))
- (set! tech-pull-channel-portions
  `((req push ,width-one)
  (ack pull ,width-one)
  (data pull ,width-n)))

startup.scm: HCs

- Reuse existing definition:
  - (brz-add-primitive-part-implementation "Adapt"
    '(style "sync"
      (include style "four_b_rb" "Adapt")))
- Write a new one:
  - (brz-add-primitive-part-implementation "Fork"
    '(style "sync"
      ...))

ABS

- Describes ‘ABStract’ gates from which a component is composed
- Lispy syntax with a Scheme expression sub-language
- Makes simple iterations simple
- Supplemented by macros for complicated description (BinaryFunc!)
- Read share/tech/common/components.abs, then share/style/STYLE/startup.scm

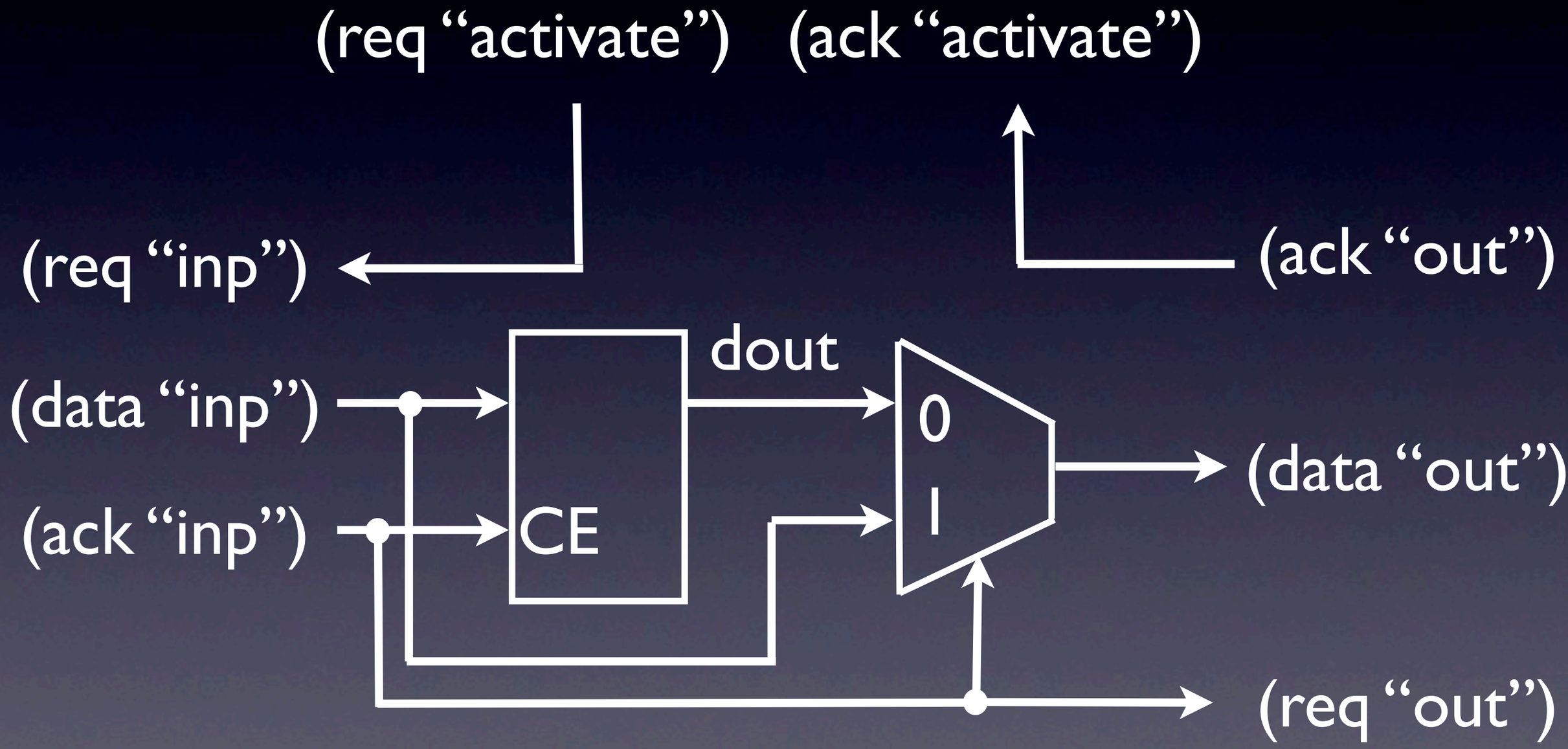
ABS HC description

- (primitive-part “name”
  (parameters (“name” type) ...)
  (ports (port “name” sense dir type)
  (symbol ...)
  (implementation ...)
)
- Standard components parameters/ports described in:
  - share/tech/common/components.abs

ABS implementation

- (style “name”
    (nodes (“name” width lowlx count) ...)
    (gates ...)
)
- Can insert an implementation into existing component with:
  brz-add-primitive-part-implementation in STYLE/startup.scm

Synchronous Fetch



Fetch in startup.scm

- (nodes
    ("dout" (param "width") 0 1)
)
- (gates
    (cell "d-flip-flop-clocked" (node "dout")
      (data "inp") (combine (dup (param "width")
        (ack "inp"))))
    (mux2 (data "out") (node "dout") (data "inp")
      (combine (dup (param "width") (ack "inp"))))
    (connect (ack "inp") (req "out"))
    (connect (ack "out") (ack "activate"))
    (connect (req "activate") (req "inp"))
)

One gate

- (cell "d-flip-flop-clocked"
    (node "dout")
    (data "inp")
    (combine (dup (param "width") (ack "inp")))
)
- Will produce (param “width”) DFFs
  - foreach i in [0..7]:
      DFF (dout_0n[i], inp_0d[i], inp_0a)

Connections

- PORTION-NAME ::= req | ack | data | node ...
- CONNECTIONS ::= CONNECTIONS +
  | (combine CONNECTIONS)
  | (PORTION-NAME VECTOR-ELEMS)
  | (smash CONNECTIONS)
- VECTOR-ELEMS ::= VECTOR-ELEM
  | (each “NAME”)
  | (dup COUNT VECTOR-ELEMS)
  | (dup-each COUNT VECTOR-ELEMS)
- VECTOR-ELEM ::= “NAME”
  | (bundle “NAME” INDEX)
  | (slice OFFSET WIDTH VECTOR-ELEM) ...
- (combine (dup (param "width") (ack "inp")))

Balsa tech. mapping

- Pretty simple/naïve. Progresses:
  - ABS gates (and (node “a”) ...)
  - Simple gates - fixed fanin:
    (“and2” (“a_0n” 0 8) ...)
  - Tech gates - mapped to tech.:
    (“q1and2d1” (“a_0n” 0 8) ...)
- Naming used to be netlist format specific.
  Verilog remains the only supported format

SSEM test harness

- Limitation: test harness generation is not easily configurable
- (set! tech-single-rail-styles
  `("sync" ,@tech-single-rail-styles))
- balsa-make-impl-test.scm modified!

Helper cells

- sr-sd-ao-ff-clocked
  - Set Rest Set Dominant Flip-flop Clocked
  - Need to describe for each technology
  - Will's make-technology script will fix this requirement

Subverting balsa-netlist

- Breeze optimisation + new components
- call-scheme from ABS expression language
- Modify balsa-netlist to inject netlists
  - maybe using Simple gates/ABS gates
- Modify generated netlists