Balsa Language Essentials

Control Flow and Commands

command	Notes
sync	Control only (dataless) handshake
<-	handshake data transfer from an expression to an output port
->	handshake data transfer to a variable from an input port
:=	assigns a value to a variable
;	sequence operator
||	parallel composition operator
continue	a null command
halt	causes deadlock
loop … end	repeat forever
while … else … end	conditional loop
for … in … then … end	structural (not temporal) iteration
if … then … else … end	conditional execution, may have multiple guarded commands
case … of … end	conditional execution based on constant expressions
select … end	non-arbitrated choice operator
arbitrate … end	arbitrated choice operator
print	compile time printing of diagnostics

Balsa Binary Operators

Symbol	Operation	Valid types	Notes
.	record indexing	record	
#	smash	any	takes value from any type and reduces it to an array of bits
[]	array indexing	array	non-const index possible, can generate lots of hardware
.	record indexing	record	
^	exponentiation	numeric	
not, log, - (unary)	unary operators	numeric	log only works on constants and returns the ceiling: e.g. log 15 returns 4 - returns a result 1 bit wider than the argument
^	exponentiation	numeric	

*, /, %	multiply, divide, remainder	numeric	only applicable to constants
+,-	add, subtract	numeric	results are one or 2 bits longer than the largest argument
@	concatenation	arrays	
<, >, <=, >=	inequalities	numeric, enumerations	
=, /=	equals, not equals	all	comparison is by sign extended value for signed numeric types
and	bitwise AND	numeric	Balsa uses type `1 bits` for `if/` `while` guards so bitwise and logical operators are the same.
or xor	bitwise OR, XOR	numeric	

Balsa Design File Organisation

<table>
<tr><td><file></td><td>::=</td><td><imports><declarations></td></tr>
<tr><td><imports></td><td>::=</td><td>(import [<path>])*</td></tr>
<tr><td><declarations></td><td>::=</td><td>(<qualifier>?<declaration>)*</td></tr>
<tr><td><qualifier></td><td>::=
|</td><td>private
public</td></tr>
<tr><td><declaration></td><td>::=
|
|</td><td>type <identifier> is <type_decl_body>
constant <identifier> = <expression> (: <type>)?
procedure <identifier> ((<formal_ports>))? is <block></td></tr>
</table>

Procedure Structure

<table>
<tr><td><declaration></td><td>::=
|</td><td>…
procedure <identifier> ((<formal_ports>))? is <block></td></tr>
<tr><td><formal_ports></td><td>::=</td><td><port> (; <port>)*</td></tr>
<tr><td><port></td><td>::=
|
|
|</td><td>(input | output) <identifiers> : type
array <range> of (input | output) <identifiers> : type
sync <identifiers>
array <range> of sync <identifiers></td></tr>
<tr><td><identifiers></td><td>::=</td><td><identifier> (, <identifier>)*</td></tr>
<tr><td><block></td><td>::=
|</td><td>(local <local_declarations>)? begin <command> end
(<command>)</td></tr>
<tr><td><local_declarations></td><td>::=</td><td>(<local_declaration>)*</td></tr>
<tr><td><local_declaration></td><td>::=
|
|
|
|
|</td><td>(declaration)
variable <identifiers> : <type>
channel <identifiers> : <type>
array <range> of channel <identifiers> : <type>
sync <identifiers>
shared <identifiers> is <block></td></tr>
</table>