Quartz Language Reference Card

conventions used in reference card

$\sigma, \sigma_1, \sigma_2$	boolean expressions
τ, τ_i, π	general expressions
λ, λ_i	left-hand side (lhs) expressions
n, m	compile-time constant expressions
α_1, α_2	data types
ℓ, ℓ_1, ℓ_2	control flow locations

module import and implemenation

package pntName	pntName like dir1.dir2.dir3 is a suffix of current dir.'s path; the remaining prefix is the root path
import pntName	pntName is added to the root path and refers then to a called module
include pntName	include textfile: (cf. **import**)
macro $f(x_1, \ldots, x_n) = \tau$	macro expression definition
// *comment*	single line comment
/* *comment* */	block comment (mult. lines)
module *m*(*vdcl*){ *stat* } [*task* list]	module m with variable declarations *vdcl*, body statement *stat* and optional task list

variable declarations *vdcl*::=

general syntax is a comma-separated list of single declarations
[storage] type [flow] $x_1, \ldots, x_n$

storage *storage*::=

mem	memorized variable (store last values)
event	event variable (reset to default value)
clocked	clocked variable (not always present)
hybrid	hybrid variable (discr.& cont. beh.)

data types *type*::=

bool	booleans
nat	unbounded unsigned integers
nat$\{n\}$	integers in $\{0, \ldots, n-1\}$
int	unbounded signed integers
int$\{n\}$	integers in $\{-n, \ldots, 0, \ldots, n-1\}$
real	real numbers
bv	unbounded bitvectors
bv$\{n\}$	bitvector of length n
[n]α	array having n elements of type α
$\alpha_1 * \ldots * \alpha_n$	tuple type

information flow *flow*::=

?	input variable (only readable)
!	output variable (only writable)
	inout variable (readable and writable)

task declarations *task*::=

driver for simulations

drivenby [*name*] { *stat* }	simulation with stimuli generator *stat* (writing inputs, reading outputs)

specs for verification

satisfies [*name*] { [*obs*] [(*goal*) list] }	verification using optional observer and proof goals
observer(*vdcl*){ *stat* }	observer with local declarations *vdcl* and body statement *stat*

expressions

constants

false	boolean constant false
true	boolean constants true

type conversions

nat2bv(τ, n)	convert **nat** to n-bit radix-2 number
int2bv(τ, n)	convert **int** to n-bit 2-complement
arr2bv(x)	convert boolean array x to bitvector
tup2bv(τ)	convert boolean tuple to bitvector
bv2nat(τ)	interpret bitvector as radix-2 number
bv2int(τ)	interpret bitvector as 2-complement num.
nat2real(τ)	convert **nat** to real number
int2real(τ)	convert **int** to real number
ceil(τ)	convert **real** to next greater **int**
floor(τ)	convert **real** to next smaller **int**

bitvector operations

$\tau\{n\}$	bit τ_n of bitvector $\tau = (\tau_\ell, \ldots, \tau_0)$
$\tau\{m{:}n\}$	segment $\tau_m \ldots \tau_n$ (with $m \geq n$)
$\tau\{m{:}\}$	segment $\tau_m \ldots \tau_0$ (with $m \geq 0$)
$\tau\{{:}n\}$	segment $(\tau_\ell, \ldots, \tau_n)$ (with $\ell \geq n$)
reverse(τ)	reverse bitvector
τ_1 @ τ_2	bitvector concatenation
$\{\tau{::}n\}$	concatenate n instances of boolean τ

constructing and accessing compound types

$\tau[\pi]$	array access
$[\tau_0, \ldots, \tau_n]$	array of $n+1$ values
$\tau.n$	tuple access
$(\tau_0, \ldots, \tau_n)$	tuple of $n+1$ values

misc. expressions

$(\tau ? \tau_1 : \tau_0)$	if τ then τ_1 else τ_0
next(τ)	value of τ in next step
$f(\tau_1, \ldots, \tau_n)$	macro function application

equality

τ_1 == τ_2	equality
τ_1 != τ_2	inequality

numeric relations

$\tau_1 < \tau_2$	less than
$\tau_1 <= \tau_2$	less than or equal to
$\tau_1 > \tau_2$	greater than
$\tau_1 >= \tau_2$	greater than or equal to

boolean operators

! σ	**not** σ	negation
σ_1 & σ_2	σ_1 **and** σ_2	conjunction
$\sigma_1 \mid \sigma_2$	σ_1 **or** σ_2	disjunction
σ_1 ^ σ_2	σ_1 **xor** σ_2	exclusive or
σ_1 -> σ_2	σ_1 **imp** σ_2	implication
σ_1 <-> σ_2	σ_1 **eqv** σ_2	equivalence

arithmetic operators

$+\pi$	unary plus (converts **nat** to type **int**)
$-\pi$	unary minus
$\tau + \pi$	addition
$\tau - \pi$	subtraction
$\tau * \pi$	multiplication
τ / π	division
τ % π	modulo
abs(τ)	absolute value
sat$\{n\}(\tau)$	saturate τ to type **nat**$\{n\}$ or **int**$\{n\}$ depending of τ's type

other operators

sin(π)	sinus
cos(π)	cosinus
exp(τ, π)	τ^π
log(π)	logarithm to base 2 (for π:**nat**, it is $\lceil \log_2(\pi) \rceil$)
sizeOf(π)	

generic expressions

exists $(i = m..n)\ \sigma_i$	denotes $\bigvee_{i=m}^{n} \sigma_i$
forall $(i = m..n)\ \sigma_i$	denotes $\bigwedge_{i=m}^{n} \sigma_i$
sum $(i = m..n)\ \tau_i$	denotes $\sum_{i=m}^{n} \tau_i$

clocked systems

clk(λ)	clock of lhs-expression λ

hybrid systems

drv(τ)	derivation of τ by physical time
cont(τ)	switch between continuous and discrete value
time	physical time for hybrid systems

Quartz Language Reference Card

statements *stat*::=

discrete statements

discrete actions

$\lambda = \tau$	immediate assignment
next(λ) = τ;	delayed assignment
emit(λ);	immediate emission
emit next(λ);	delayed emission
[*name* :] **assume**(σ);	assumption
[*name* :] **assert**(σ);	assertion

wait statements

nothing;	empty statement
[ℓ :] **pause**;	separate macro steps
[ℓ :] **halt**;	halt forever
[ℓ :] **[immediate] await** (σ);	
	wait until σ holds

conditional statements

if (σ) S_1 [**else** S_2]	
	if σ holds, execute S_1 otherwise S_2
choose S_1 **else** S_2	
	nondeterministic choice
case	equivalent to
(σ_1) **do** S_1	**if** (σ_1) S_1
(σ_2) **do** S_2	**else if** (σ_2) S_2
...	...
(σ_n) **do** S_n	**else if** (σ_n) S_n
default S_{n+1}	**else** S_{n+1}

sequential and parallel control flow

S_1 S_2	sequential execution
S_1 || S_2	synchronous |-parallel
S_1 ||| S_2	asynchronous |-parallel
S_1 | S_2	interleaved |-parallel
S_1 && S_2	synchronous &-parallel
S_1 &&& S_2	asynchronous &-parallel
S_1 & S_2	interleaved &-parallel

loops

loop S	infinite loop of S
do S **while**(σ)	repeat S while σ holds
while (σ)S	while σ holds, repeat S
always S	infinite loop of **pause**;S;
immediate always S	
	infinite loop of S;**pause**;

local declarations

{ α x; S }	declare variable x of type α with scope S
let ($x = \tau$) S	abbreviate τ by x in S

generic statements (will be unrolled)

for (i=m .. n) S	generic sequence
for (i=m .. n) **do** η S	
	generic parallel with $\eta \in$ {|,&,||,&&,|||,&&& }
choose (i=m .. n) S	generic nondeterministic choice

module call

[iname:] $m(\tau_1, \ldots, \tau_n)$;

means: instance *iname* of call to module m;

- inputs of m must be readable expressions τ_i
- outputs of m must be writable lhs-expressions τ_i
- undesired outputs of m can be skipped by _

abortion, suspension and during statements

[weak] [immediate] abort S **when**(σ);
aborts S when σ holds

[weak] [immediate] suspend S **when**(σ);
suspends S when σ holds

[immediate] [final] during S_1 **do** S_2;
in each step of S_1 do also instantaneous S_2

hybrid systems statements *stat*::=

(generic) flow statements

flow[(i=m .. n)] {S_1; ... ; S_n}	perform continuous actions S_i until interrupted
flow[(i=m .. n)]{ S_1; ... ; S_n } **until**(σ);	perform continuous actions S_i until σ holds

continuous actions

x <− τ	continuous assignment
drv(x) <− τ;	derivative assignment
[*name* :] constrainSME	continuous assertion with at least one of S, M, E

proof goals *goal*::=

assumption

name: **assume** *spec*;

assertion goal

name [*vtask*] {*cl*}: **assert** *spec* [**with** {*al*}];

- *cl* is the list of controllable variables
- *al* is the list of assumptions

verification task *vtask*::=

ProveE	property is true in one initial state
ProveA	property is true in all initial states
DisProveE	property is false in one initial state
DisProveA	property is false in all initial states

specifications *spec*::=

path quantifiers

A φ	φ holds on all infinite computation paths
E φ	φ holds on one infinite computation path

linear time future operators

X φ	φ holds in the next point
G φ	always φ in the future
F φ	eventual φ in the future
[φ **SU** ψ]	φ until ψ holds and ψ must hold
[φ **SB** ψ]	φ before ψ holds and φ must hold
[φ **SW** ψ]	φ when first ψ holds and ψ must hold
[φ **WU** ψ]	φ until ψ holds or φ holds forever
[φ **WB** ψ]	φ before ψ holds or ψ never holds
[φ **WW** ψ]	φ when first ψ holds or ψ never holds

linear time past operators

PSX φ	φ holds in the previous point and there is a previous point
PWX φ	φ holds in the previous point or no previous point
PG,**PF** φ	past time **G**,**F**
PSU,**PSB**,**PSW**	past time **SU**,**SB**,**SW**
PWU,**PWB**,**PWW**	past time **WU**,**WB**,**WW**

mu calculus operators

nu z. φ	greatest fixpoint wrt. z
mu z. φ	least fixpoint wrt. z
<> φ	φ holds in one successor state
[] φ	φ holds in all successor states
<:> φ	φ holds in one predecessor state
[:] φ	φ holds in all predecessor states