Dax Phyz Logics Reference

SYMBOLS

obj	O, O
vert	V, V
rod	r, R
stick	s, S
anchor	A, A
pin	P, P
spring	S, SP
hslot	h, HS
vslot	v, VS
rocket	R, RT
charge	C, C
heat	H, H
logic	L, L

PHYSICS FRAME EXECUTION ORDER

1: Edge triggered logics are executed once.
2: PC-clocked logics' enabled states are calculated and set.
3: Enabled non-edge and PC-triggered logics are executed once.
4: Enabled non-edge logics and enabled PC-triggered logics with edge<0 are executed continously during the physics frame.

A PC-clocked logic is enabled both when its edge==PC and when its edge==-PC (i.e. if edge is negative). If edge>0, the logic behaves as a normal edge triggered logic, and thus is executed exactly once in the physics frame. If edge<0, the logic behaves as a non-edge logic and is executed continously in the physics frame.

LOGIC TYPES

category	**name**	**ppp P O A ttt** **123 C X E 123**	**mandat** **123**	**p1**	**p2**	**p3**	**operation desc** (e): inherently edge triggered
anchor	AVR	VL- L - 1 -A-	-L-	[V]	LA		(e) V = LA.A.V
anchor	AXR	L-- L v 1 A--	L--	LA			(e) v = LA.A.x
anchor	AXW	LL- L - 1 A--	LL-	LA	Lx		(e) LA.A.x = Lx.v
anchor	AYR	L-- L v 1 A--	L--	LA			(e) v = LA.A.y
anchor	AYW	LL- L - 1 A--	LL-	LA	Ly		(e) LA.A.y = Ly.v
assign	CPY	LL- L - - ---	LL-	Ld	Ls		Ld.v = Ls.v
assign	CPYIF	LLL L - - ---	LLL	Ld	Ls	Lb	IF Lb.v THEN Ld.v=Ls.v
assign	DEC	L-- L - 1 ---	L--	L			(e) L.v = L.v - 1
assign	INC	L-- L - 1 ---	L--	L			(e) L.v = L.v + 1
assign	INV	L-- L - 1 ---	L--	L			(e) L.v = !(L.v)
charge	CCR	L-- L v 1 C--	L--	LC			(e) v = LC.C.charge
charge	CCW	LL- L - 1 C--	LL-	LC	Lc		(e) LC.C.charge = Lc.v
charge	CVR	VL- L - 1 -C-	-L-	[V]	LC		(e) V = LC.C.V
del	DELA	L-- L - 1 A--	L--	LA			(e) delete anchor LA.A
del	DELC	L-- L - 1 C--	L--	LC			(e) delete charge LC.C
del	DELH	L-- L - 1 H--	L--	LH			(e) delete heat LH.H
del	DELHS	L-- L - 1 h--	L--	Lh			(e) delete hslot Lh.h
del	DELL	L-- L - 1 L--	L--	LL			(e) delete logic LL.L1
del	DELO	L-- L - 1 O--	L--	LO			(e) delete obj LO.O
del	DELP	L-- L - 1 P--	L--	LP			(e) delete pin LP.P
del	DELR	L-- L - 1 r--	L--	Lr			(e) delete rod Lr.r
del	DELRT	L-- L - 1 R--	L--	LR			(e) delete rocket LR.R
del	DELS	L-- L - 1 s--	L--	Ls			(e) delete stick Ls.s
del	DELSP	L-- L - 1 S--	L--	LS			(e) delete spring LS.S
del	DELV	L-- L - 1 V--	L--	LV			(e) delete vert LV.V
del	DELVS	L-- L - 1 v--	L--	Lv			(e) delete vslot Lv.vs
env	DBQRY	VL- L - - ---	V--	Vcon	[Ldirty]		{no operation (db conn)}
env	DBQRZ	LLL L - 1 VVV	LLL	Ldbq	LVqry	LVres	(e) {db query/store}
env	PF	--- L v 1 ---	---				(e) v = {cur phyz frame}
env	POSXR	--- L v 1 ---	---				(e) v = {cur scr x-pos}
env	POSXW	L-- L - 1 ---	L--	Lx			(e) {cur scr x-pos} = Lx.v
env	POSYR	--- L v 1 ---	---				(e) v = {cur scr y-pos}
env	POSYW	L-- L - 1 ---	L--	Ly			(e) {cur scr y-pos} = Ly.v
env	PRINT	--- L - 1 ---	---				(e) {print}
env	SMSG	LLL L - 1 ---	LLL	Lmsg	Lwparam	Llparam	(e) v = {sendmsg retval}
env	SOUND	LL- L - 1 ---	LL-	Lsound	Lvol		(e) {play sound}
env	STOP	--- L - 1 ---	---				(e) stop physics
env	ZOOMR	--- L v 1 ---	---				(e) v = {cur scr zoom}
env	ZOOMW	L-- L - 1 ---	L--	Lz			(e) {cur scr zoom} = Lz.v
func	ABS	L-- L v - ---	L--	L			v = abs(L.v)
func	ADD	LL- L v - ---	LL-	La	Lb		v = La.v + Lb.v
func	AND	LL- L v - ---	LL-	La	Lb		v = (La.v) && (Lb.v)
func	DIV	LL- L v - ---	LL-	La	Lb		v = La.v / Lb.v
func	EOR	LL- L v - ---	LL-	La	Lb		v = (La.v) ^^ (Lb.v)

func	EQ	LL- L v - ---	LL-	La	Lb		v = (La.v == Lb.v)
func	EXP	L-- L v - ---	L--	L			v = exp(L.v)
func	GEQ	LL- L v - ---	LL-	La	Lb		v = (La.v >= lb.v)
func	GT	LL- L v - ---	LL-	La	Lb		v = (La.v > Lb.v)
func	IF	LLL L v - ---	LLL	Lb	Lc	La	v = (Lb.v) ? Lc.v : La.v
func	LEQ	LL- L v - ---	LL-	La	Lb		v = (La.v <= Lb.v)
func	LOG	L-- L v - ---	L--	L			v = log(L.v)
func	LT	LL- L v - ---	LL-	La	Lb		v = (La.v < Lb.v)
func	MAX	LL- L v - ---	LL-	La	Lb		v = max(La.v, Lb.v)
func	MIN	LL- L v - ---	LL-	La	Lb		v = min(La.v, Lb.v)
func	MUL	LL- L v - ---	LL-	La	Lb		v = La.v * Lb.v
func	NEQ	LL- L v - ---	LL-	La	Lb		v = (La.v != Lb.v)
func	NOT	L-- L v - ---	L--	L			v = !(L.v)
func	OR	LL- L v - ---	LL-	La	Lb		v = (La.v) || (Lb.v)
func	RAND	--- L v - ---	---				v = {random float}
func	REG	VLL L - - ---	---	[V]	[L1]	[L2]	no operation
func	RGB	LLL L v - ---	LLL	Lr	Lg	Lb	v = {RGB val}
func	SUB	LL- L v - ---	LL-	La	Lb		v = La.v - Lb.v
func	TRUNC	L-- L v - ---	L--	L			v = trunc(L.v)
generic	XCPY	LLL L - 1 ---	LL-	Ld	Ls	[Ldir]	(e) Ld.{X} = Ls.{X}
generic	XEQ	LL- L v 1 ---	LL-	La	Lb		(e) v = (La.{X} == Lb.{X})
generic	XNUL	L-- L v 1 ---	L--	L			(e) v = (L.{X} == -1})
heat	HHR	L-- L v 1 H--	L--	LH			(e) v = LH.H.heat
heat	HHW	LL- L - 1 H--	LL-	LH	L		(e) LH.H.heat = L.v
heat	HVR	VL- L - 1 -H-	-L-	[V]	LH		(e) V = LH.H.V
hslot	HSV1R	VL- L - 1 -h-	-L-	[V]	Lh		(e) V = Lh.h.V1
hslot	HSV1W	LL- L - 1 hV-	LL-	Lh	LV		(e) Lh.h.V1 = LV.V
hslot	HSV2R	VL- L - 1 -h-	-L-	[V]	Lh		(e) V = Lh.h.V2
hslot	HSV2W	LL- L - 1 hV-	LL-	Lh	LV		(e) Lh.h.V2 = LV.V
hslot	HSVR	VL- L - 1 -h-	-L-	[V]	Lh		(e) V = Lh.h.V
hslot	HSYR	L-- L v 1 h--	L--	Lh			(e) v = Lh.h.y
hslot	HSYW	LL- L - 1 h--	LL-	Lh	Ly		(e) Lh.h.y = Ly.v
keyboard	KEYS1	L-- L - 1 ---	L--	Lbool			(e) en/disable 1-kb shortcuts
keyboard	KEYS2	L-- L - 1 ---	L--	Lbool			(e) en/disable n-kb shortcuts
keyboard	KPBC	--- L - 1 ---	---				(e) clears keypress buffer
keyboard	KPBR	VLL L v 1 ---	---	[Vundo]	[Lfocus]	[Ledit]	(e) v = {buffered keypress}
keyboard	KSR	L-- L v 1 ---	L--	Lkey			(e) v = {key state of Lkey}
logic	LEDR	L-- L v 1 L--	L--	LL			(e) v = LL.L.edge
logic	LEDW	LL- L - 1 L--	LL-	LL	Le		(e) LL.L.edge = Le.v
logic	LENR	L-- L v 1 ---	L--	L			(e) v = L.enabled
logic	LENW	LL- L - 1 ---	LL-	L	Lbool		(e) L.enabled = Lbool.v
logic	LL1R	LL- L - 1 -L-	-L-	[L]	LL		(e) L = LL.L.L1
logic	LL1W	LL- L - 1 LL-	LL-	LL	LL1		(e) LL.L.L1 = LL1.L1
logic	LL2R	LL- L - 1 -L-	-L-	[L]	LL		(e) L = LL.L.L2
logic	LL2W	LL- L - 1 LL-	LL-	LL	LL2		(e) LL.L.L2 = LL2.L1
logic	LL3R	LL- L - 1 -L-	-L-	[L]	LL		(e) L = LL.L.L3
logic	LL3W	LL- L - 1 LL-	LL-	LL	LL3		(e) LL.L.L3 = LL3.L1
logic	LL4R	LL- L - 1 -L-	-L-	[L]	LL		(e) L = LL.L.L4
logic	LL4W	LL- L - 1 LL-	LL-	LL	LL4		(e) LL.L.L4 = LL4.L1
logic	LTR	L-- L v 1 L--	L--	LL			(e) v = LL.L.type
logic	LTW	LL- L - 1 L--	LL-	LL	Lt		(e) LL.L.type = Lt.v
logic	LXR	L-- L v 1 L--	L--	LL			(e) v = LL.L.x
logic	LXW	LL- L - 1 L--	LL-	LL	Lx		(e) LL.L.x = Lx.v
logic	LYR	L-- L v 1 L--	L--	LL			(e) v = LL.L.y
logic	LYW	LL- L - 1 L--	LL-	LL	Ly		(e) LL.L.y = Ly.v
logic	LZR	L-- L v 1 L--	L--	LL			(e) v = LL.L.z
logic	LZW	LL- L - 1 L--	LL-	LL	Lz		(e) LL.L.z = Lz.v
mouse	MCBC	--- L - 1 ---	---				(e) clears mouse clk buffer
mouse	MCBR	L-- L v 1 ---	---	[L]			(e) v = {buffered mouse clk}
mouse	MCBRX	--- L v 1 ---	---				(e) v = {buf mouse clk x-pos}
mouse	MCBRY	--- L v 1 ---	---				(e) v = {buf mouse clk y-pos}
mouse	MPXR	--- L v 1 ---	---				(e) v = {cur mouse x-pos}
mouse	MPYR	--- L v 1 ---	---				(e) v = {cur mouse y-pos}
new	NEWA	AL- L - 1 -V-	-L-	[A]	LV		(e) A = {new anchor}
new	NEWC	CL- L - 1 -V-	-L-	[C]	LV		(e) C = {new charge}
new	NEWH	HL- L - 1 -V-	-L-	[H]	LV		(e) H = {new heat}
new	NEWHS	hL- L - 1 -V-	-L-	[h]	LV		(e) h = {new hslot}

new	NEWL	LLL L - 1 ---	-LL	[L]	Lx	Ly	(e) L = {new logic}
new	NEWP	PLL L - 1 -VV	-LL	[P]	LV1	LV2	(e) P = {new pin}
new	NEWR	rLL L - 1 -VV	-LL	[r]	LV1	LV2	(e) r = {new rod}
new	NEWRT	RL- L - 1 -V-	-L-	[R]	LV		(e) R = {new rocket}
new	NEWS	sLL L - 1 -VV	-LL	[s]	LV1	LV2	(e) s = {new stick}
new	NEWSP	SLL L - 1 -VV	-LL	[S]	LV1	LV2	(e) S = {new spring}
new	NEWV	VLL L - 1 ---	-LL	[V]	Lx	Ly	(e) V = {new vert}
new	NEWVS	hL- L - 1 -V-	-L-	[vs]	LV		(e) vs = {new vslot}
object	OCOLR	L-- L v 1 O--	L--	LO			(e) v = LO.O.col
object	OCOLW	LL- L - 1 O--	LL-	LO	Lcol		(e) LO.O.col = Lcol.v
object	OLR	L-- L v 1 O--	L--	LO			(e) v = LO.O.level
object	OLW	LL- L - 1 O--	LL-	LO	Llvl		(e) LO.O.level = Llvl.v
object	OVXR	L-- L v 1 O--	L--	LO			(e) v = LO.O.vx
object	OVXW	LL- L - 1 O--	LL-	LO	Lvx		(e) LO.O.vx = Lvx.v
object	OVYR	L-- L v 1 O--	L--	LO			(e) v = LO.O.vy
object	OVYW	LL- L - 1 O--	LL-	LO	Lvy		(e) LO.O.vy = Lvy.v
object	OXR	L-- L v 1 O--	L--	LO			(e) v = LO.O.x
object	OXW	LL- L - 1 O--	LL-	LO	Lx		(e) LO.O.x = Lx.v
object	OYR	L-- L v 1 O--	L--	LO			(e) v = LO.O.y
object	OYW	LL- L - 1 O--	LL-	LO	Ly		(e) LO.O.y = Ly.v
pin	PV1R	VL- L - 1 -P-	-L-	[V]	LP		(e) V = LP.P.V1
pin	PV2R	VL- L - 1 -P-	-L-	[V]	LP		(e) V = LP.P.V2
rocket	RTAR	L-- L v 1 R--	L--	LR			(e) v = LR.R.angle
rocket	RTAW	LL- L - 1 R--	LL-	LR	La		(e) LR.R.angle = La.v
rocket	RTFR	L-- L v 1 R--	L--	LR			(e) v = LR.R.force
rocket	RTFW	LL- L - 1 R--	LL-	LR	Lf		(e) LR.R.force = Lf.v
rocket	RTVR	VL- L - 1 -R-	-L-	[V]	LR		(e) V = LR.R.V
rocket	RTVRR	VL- L - 1 -R-	-L-	[V]	LR		(e) V = LR.R.VR
rocket	RTVRW	LL- L - 1 RV-	LL-	LR	LV		(e) LR.R.VR = LV.V
rod	RLR	L-- L v 1 r--	L--	Lr			(e) v = Lr.r.len
rod	RLW	LL- L - 1 r--	LL-	Lr	Llen		(e) Lr.r.len = Llen.v
rod	RV1R	VL- L - 1 -r-	-L-	[V]	Lr		(e) V = Lr.r.V1
rod	RV2R	VL- L - 1 -r-	-L-	[V]	Lr		(e) V = Lr.r.V2
spring	SPAR	L-- L v 1 S--	L--	LS			(e) v = LS.S.a
spring	SPAW	LL- L - 1 S--	LL-	LS	La		(e) LS.S.a = La.v
spring	SPBR	L-- L v 1 S--	L--	LS			(e) v = LS.S.b
spring	SPBW	LL- L - 1 S--	LL-	LS	Lb		(e) LS.S.b = Lb.v
spring	SPCLR	L-- L v 1 S--	L--	LS			(e) v = LS.S.curlen
spring	SPDR	L-- L v 1 S--	L--	LS			(e) v = LS.S.k
spring	SPDW	LL- L - 1 S--	LL-	LS	Ld		(e) LS.S.d = Ld.v
spring	SPFR	L-- L v 1 S--	L--	LS			(e) v = LS.S.curforce
spring	SPKR	L-- L v 1 S--	L--	LS			(e) v = LS.S.k
spring	SPKW	LL- L - 1 S--	LL-	LS	Lk		(e) LS.S.k = Lk.v
spring	SPLR	L-- L v 1 S--	L--	LS			(e) v = LS.S.len
spring	SPLW	LL- L - 1 S--	LL-	LS	Llen		(e) LS.S.len = Llen.v
spring	SPSR	SL- L - 1 -S-	-L-	[S]	LS		(e) S = LS.S.S
spring	SPSW	LL- L - 1 SS-	LL-	LS	LSS		(e) LS.S.S = LSS.S
spring	SPV1R	VL- L - 1 -S-	-L-	[V]	LS		(e) V = LS.S.V1
spring	SPV2R	VL- L - 1 -S-	-L-	[V]	LS		(e) V = LS.S.V2
sprite	EDBOX	VLL L - 1 ---	VL-	Veb	Lkp	[Lextra]	(e) {handle editbox Veb}
sprite	EDCLK	L-- L - 1 ---	L--	Ledbox			(e) {click button Ledbox}
sprite	VTRC	LL- L - 1 V--	L--	LV	[Lpos]		(e) v = {LV.V.sprite char}
sprite	VTRL	L-- L - 1 V--	L--	LV			(e) v = {LV.V.sprite len}
sprite	VTWB	LLL L - 1 V--	LLL	LV	Lb	Lpos	(e) LV.V.sprite = {Lb.v byte}
sprite	VTWC	LLL L - 1 V--	LL-	LV	Lchr	[Lpos]	(e) LV.V.sprite = {Lchr.v char}
sprite	VTWF	LLL L - 1 V--	LL-	LV	Lf	[Lprec]	(e) LV.V.sprite = {Lf.v float}
stick	SLR	L-- L v 1 s--	L--	Ls			(e) v = Ls.s.len
stick	SLW	LL- L - 1 s--	LL-	Ls	Llen		(e) Ls.s.len = Llen.v
stick	SSR	L-- L v 1 s--	L--	Ls			(e) v = Ls.s.str
stick	SSW	LL- L - 1 s--	LL-	Ls	Lstr		(e) Ls.s.str = Lstr.v
stick	SV1R	VL- L - 1 -s-	-L-	[V]	Ls		(e) V = Ls.s.V1
stick	SV2R	VL- L - 1 -s-	-L-	[V]	Ls		(e) V = Ls.s.V2
stick	SXR	L-- L v 1 s--	L--	Ls			(e) v = Ls.s.expl
stick	SXW	LL- L - 1 s--	LL-	Ls	Lx		(e) Ls.s.expl = Lx.v
vert	VCOLR	L-- L v 1 V--	L--	LV			(e) v = LV.V.col
vert	VCOLW	LL- L - 1 V--	LL-	LV	Lcol		(e) LV.V.col = Lcol.v
vert	VDIST	LL- L v 1 VV-	LL-	LV1	LV2		(e) v = dist(LV1.V, LV2.V)

vert	VOR	OL- L - 1 -V-	-L-	[O]	LV		(e) O = LV.V.O
vert	VVXR	L-- L v 1 V--	L--	LV			(e) v = LV.V.vx
vert	VVXW	LL- L - 1 V--	LL-	LV	Lvx		(e) LV.V.vx = Lvx.v
vert	VVYR	L-- L v 1 V--	L--	LV			(e) v = LV.V.vy
vert	VVYW	LL- L - 1 V--	LL-	LV	Lvy		(e) LV.V.vy = Lvy.v
vert	VXR	L-- L v 1 V--	L--	LV			(e) v = LV.V.x
vert	VXW	LL- L - 1 V--	LL-	LV	Lx		(e) LV.V.x = Lx.v
vert	VYR	L-- L v 1 V--	L--	LV			(e) v = LV.V.y
vert	VYW	LL- L - 1 V--	LL-	LV	Ly		(e) LV.V.y = Ly.v
vert	VZR	L-- L v 1 V--	L--	LV			(e) v = LV.V.z
vert	VZW	LL- L - 1 V--	LL-	LV	Lz		(e) LV.V.z = Lz.v
vslot	VSVR	VL- L - 1 -v-	-L-	[V]	Lv		(e) V = Lv.vs.V
vslot	VSXR	L-- L v 1 v--	L--	Lv			(e) v = Lv.vs.x
vslot	VSXW	LL- L - 1 v--	LL-	Lv	Lx		(e) Lv.vs.x = Lx.v