Iambic Keyer: A Z Specification

Formal Model of Automatic Morse Code Element Generation

February 2026

Contents

1 Introduction

This specification models an *iambic keyer*—an electronic device that generates properly-timed Morse code elements from paddle inputs. Unlike a straight key where the operator manually times each dit and dah by holding down the key for the correct duration, an iambic keyer handles all timing automatically. The operator simply touches the dit paddle to produce dits or the dah paddle to produce dahs; the keyer ensures correct element duration and inter-element spacing.

1.1 Integration with Koch Trainer

This specification extends the Koch Trainer system (koch_trainer.tex) by modeling the keyer subsystem used during *send* training. The keyer:

- Operates only when *radioMode* = *transmitting* (half-duplex constraint)
- Generates audio via the existing *toneActive* mechanism
- Receives paddle inputs from touch zones or keyboard (F/J keys in simulator)
- Outputs properly-timed Morse elements for character recognition

The keyer is the bridge between raw user input (paddle touches) and the discrete character attempts recorded by *RecordSendAttempt*.

1.2 Keyer Variants

Amateur radio operators use several keyer modes. This specification models *Iambic Mode B*, the most common choice:

Mode	Behavior
Iambic A	Element stops immediately when both paddles released
Iambic B	Current element completes before stopping
Ultimatic	Alternation favors most recently pressed paddle

Mode B is preferred by most operators because it produces cleaner code—the current element always completes, avoiding truncated elements that sound wrong.

1.3 Scope

This specification models:

- Paddle input state (dit/dah held)
- Keyer state machine (idle, playing element, inter-element gap)
- Element timing derived from words-per-minute setting
- Iambic squeeze behavior (alternating dit-dah when both paddles held)
- Haptic feedback triggers

We do not model audio waveform generation, envelope shaping, or UI layout—only the state machine that determines when tones start and stop.

2 Basic Types

Time is measured in milliseconds for element timing precision.

$$Time == \mathbb{N}$$

We reuse the boolean type from koch_trainer.tex:

$$ZBOOL ::= ztrue \mid zfalse$$

3 Free Types

3.1 Paddle

The two physical paddle inputs:

$$Paddle ::= dit \mid dah$$

In the physical device, dit is conventionally on the left (thumb) and dah on the right (index finger) for right-handed operators. The iPhone implementation uses the bottom-left quadrant for dit and bottom-right for dah.

3.2 Element

The Morse code elements that the keyer generates:

$$Element ::= ditElement \mid dahElement$$

A dit has duration T (one time unit). A dah has duration $3T$. The inter-element gap within a character is T. These ratios are fixed by international convention.

3.3 Keyer Phase

The keyer cycles through three phases:

$$KeyerPhase ::= keyerIdle \mid playing \mid gap$$

- *keyerIdle*: No element active, waiting for paddle input
- *playing*: Tone active, generating a dit or dah
- *gap*: Tone silent, inter-element spacing before next element

3.4 Iambic Mode

$$IambicMode ::= modeA \mid modeB$$

Mode A truncates elements on paddle release; Mode B completes them. We default to Mode B but model both for completeness.

4 Global Constants

4.1 Timing Constants

Element timing derives from the standard PARIS calibration: the word "PARIS" contains exactly 50 time units, so at W words per minute, each time unit is $\frac{1200}{W}$ milliseconds.

$$\begin{array}{l} defaultWpm : \mathbb{N} \\ minWpm : \mathbb{N} \\ maxWpm : \mathbb{N} \\ defaultToneHz : \mathbb{N} \\ minToneHz : \mathbb{N} \\ maxToneHz : \mathbb{N} \\ \hline defaultWpm = 13 \\ minWpm = 5 \\ maxWpm = 40 \\ defaultToneHz = 700 \\ minToneHz = 400 \\ maxToneHz = 1000 \end{array}$$

The default speed of 13 WPM is a good starting point for beginners. The tone frequency of 700 Hz is comfortable for extended listening; operators typically choose frequencies between 400–1000 Hz based on personal preference and ambient noise conditions.

4.2 Timing Functions

We define timing as axiomatic functions rather than computed values, since Z does not have division. The implementation computes these as:

$$ditMs = \frac{1200}{wpm}, \quad dahMs = 3 \times ditMs, \quad gapMs = ditMs$$

$$\begin{array}{|l} ditDuration : \mathbb{N} \pfun Time \\ dahDuration : \mathbb{N} \pfun Time \\ gapDuration : \mathbb{N} \pfun Time \\ \hline \mathrm{dom}\, ditDuration = minWpm \mathinner{..} maxWpm \\ \mathrm{dom}\, dahDuration = minWpm \mathinner{..} maxWpm \\ \mathrm{dom}\, gapDuration = minWpm \mathinner{..} maxWpm \\ \forall\, w : minWpm \mathinner{..} maxWpm \bullet \\ \quad dahDuration(w) = 3 * ditDuration(w) \land \\ \quad gapDuration(w) = ditDuration(w) \end{array}$$

For reference, at common speeds:

WPM	Dit (ms)	Dah (ms)	Gap (ms)
5	240	720	240
13	92	277	92
20	60	180	60
25	48	144	48

4.3 Model Bounds

As with koch_trainer.tex, we define bounds for finite model checking:

$$\begin{array}{|l} KEYER\_MODEL\_BOUND : \mathbb{N} \\ \hline KEYER\_MODEL\_BOUND = 10 \end{array}$$

This bounds the time values to enable ProB animation.

5 Keyer Configuration

User-configurable keyer settings that remain constant during element generation:

$$\begin{array}{|l} KeyerConfig \\ \hline wpm : \mathbb{N} \\ toneHz : \mathbb{N} \\ iambicMode : IambicMode \\ hapticEnabled : ZBOOL \\ \hline wpm \geq minWpm \\ wpm \leq maxWpm \\ toneHz \geq minToneHz \\ toneHz \leq maxToneHz \end{array}$$

The *wpm* setting controls element timing. The *toneHz* setting controls audio frequency (passed to the audio engine, not modeled here). The *iambicMode* selects Mode A or B behavior. The *hapticEnabled* flag controls whether haptic feedback fires on element start.

6 Paddle Input State

Tracks which paddles are currently held down:

PaddleState

$$ditHeld : ZBOOL$$
$$dahHeld : ZBOOL$$

Both paddles can be held simultaneously—this is the "squeeze" that produces alternating dit-dah sequences in iambic mode.

7 Keyer State

The core keyer state machine:

KeyerState

$$keyerPhase : KeyerPhase$$
$$currentElement : Element$$
$$elementStartTime : Time$$
$$gapStartTime : Time$$
$$pendingOpposite : ZBOOL$$
$$lastPlayedElement : Element$$

$$keyerPhase = keyerIdle \Rightarrow elementStartTime = 0 \land gapStartTime = 0$$
$$keyerPhase = playing \Rightarrow gapStartTime = 0$$
$$keyerPhase = gap \Rightarrow elementStartTime = 0$$
$$elementStartTime \leq KEYER\_MODEL\_BOUND$$
$$gapStartTime \leq KEYER\_MODEL\_BOUND$$

The *currentElement* field indicates which element is playing (only meaningful when $keyerPhase = playing$). The *elementStartTime* records when the current element began, enabling duration checking. The *pendingOpposite* flag is set when the opposite paddle is pressed during an element, ensuring it plays next (iambic squeeze behavior). The *lastPlayedElement* remembers what just finished for alternation.

8 Output State

Observable outputs—tone generation and haptic feedback:

KeyerOutput

$$toneActive : ZBOOL$$
$$hapticPending : ZBOOL$$
$$hapticPaddle : Paddle$$

The *toneActive* field connects directly to the *toneActive* field in *TrainingSession* from koch_trainer.tex. When $toneActive = ztrue$, the audio engine generates a tone at the configured frequency.

The *hapticPending* flag indicates a haptic event should fire. The *hapticPaddle* indicates which paddle triggered it (dit and dah have subtly different haptic patterns for tactile differentiation).

9 Complete Keyer System

Keyer

$$KeyerConfig$$
$$PaddleState$$
$$KeyerState$$
$$KeyerOutput$$
$$now : Time$$

$$toneActive = ztrue \Leftrightarrow keyerPhase = playing$$
$$keyerPhase = keyerIdle \Rightarrow hapticPending = zfalse$$
$$now \leq KEYER\_MODEL\_BOUND$$

The invariant $toneActive = ztrue \Leftrightarrow keyerPhase = playing$ ensures the audio output correctly reflects the keyer state. This is the key property that the implementation must maintain.

10 Initialization

KeyerInit

$Keyer'$

$wpm' = defaultWpm$
$toneHz' = defaultToneHz$
$iambicMode' = modeB$
$hapticEnabled' = ztrue$
$ditHeld' = zfalse$
$dahHeld' = zfalse$
$keyerPhase' = keyerIdle$
$currentElement' = ditElement$
$elementStartTime' = 0$
$gapStartTime' = 0$
$pendingOpposite' = zfalse$
$lastPlayedElement' = ditElement$
$toneActive' = zfalse$
$hapticPending' = zfalse$
$hapticPaddle' = dit$
$now' = 0$

The keyer initializes to an idle state with default configuration. The *currentElement* and *lastPlayedElement* default values are arbitrary since they're only meaningful in non-idle states.

11 Input Operations

11.1 Paddle Down

User presses a paddle (touch begins or key pressed):

PaddleDown

$\Delta Keyer$
$p? : Paddle$

$(p? = dit \land ditHeld' = ztrue \land dahHeld' = dahHeld) \lor (p? = dah \land dahHeld' = ztrue \land ditHeld' = ditHeld)$
$(keyerPhase = playing \land$
$\quad ((p? = dit \land currentElement = dahElement) \lor$
$\quad (p? = dah \land currentElement = ditElement)))$
$\quad \Rightarrow pendingOpposite' = ztrue$
$\neg\, (keyerPhase = playing \land$
$\quad ((p? = dit \land currentElement = dahElement) \lor$
$\quad (p? = dah \land currentElement = ditElement)))$
$\quad \Rightarrow pendingOpposite' = pendingOpposite$
$keyerPhase' = keyerPhase$
$currentElement' = currentElement$
$elementStartTime' = elementStartTime$
$gapStartTime' = gapStartTime$
$lastPlayedElement' = lastPlayedElement$
$toneActive' = toneActive$
$hapticPending' = hapticPending$
$hapticPaddle' = hapticPaddle$
$now' = now$
$wpm' = wpm$
$toneHz' = toneHz$
$iambicMode' = iambicMode$
$hapticEnabled' = hapticEnabled$

The key behavior here is setting *pendingOpposite* when the opposite paddle is pressed during an

element. This ensures the squeeze behavior works correctly—pressing dah while a dit is playing queues the dah for immediate playback after the current element completes.

11.2 Paddle Up

User releases a paddle (touch ends or key released):

PaddleUp

$$\Delta Keyer$$
$$p? : Paddle$$

$$(p? = dit \land ditHeld' = zfalse \land dahHeld' = dahHeld) \lor (p? = dah \land dahHeld' = zfalse \land ditHeld' = ditHeld)$$
$$keyerPhase' = keyerPhase$$
$$currentElement' = currentElement$$
$$elementStartTime' = elementStartTime$$
$$gapStartTime' = gapStartTime$$
$$pendingOpposite' = pendingOpposite$$
$$lastPlayedElement' = lastPlayedElement$$
$$toneActive' = toneActive$$
$$hapticPending' = hapticPending$$
$$hapticPaddle' = hapticPaddle$$
$$now' = now$$
$$wpm' = wpm$$
$$toneHz' = toneHz$$
$$iambicMode' = iambicMode$$
$$hapticEnabled' = hapticEnabled$$

Note that releasing a paddle does not immediately stop the element in Mode B. The element completes its full duration. Mode A would add a truncation check here (modeled separately in Section 16).

12 Internal State Transitions

12.1 Start Element from Idle

When the keyer is idle and a paddle is held, start the corresponding element:

StartFromIdle

$$\Delta Keyer$$

$$keyerPhase = keyerIdle$$
$$ditHeld = ztrue \lor dahHeld = ztrue$$
$$(ditHeld = ztrue \Rightarrow$$
$$\quad currentElement' = ditElement \land hapticPaddle' = dit)$$
$$(ditHeld = zfalse \land dahHeld = ztrue \Rightarrow$$
$$\quad currentElement' = dahElement \land hapticPaddle' = dah)$$
$$keyerPhase' = playing$$
$$elementStartTime' = now$$
$$gapStartTime' = 0$$
$$pendingOpposite' = zfalse$$
$$lastPlayedElement' = lastPlayedElement$$
$$toneActive' = ztrue$$
$$(hapticEnabled = ztrue \Rightarrow hapticPending' = ztrue)$$
$$(hapticEnabled = zfalse \Rightarrow hapticPending' = zfalse)$$
$$ditHeld' = ditHeld$$
$$dahHeld' = dahHeld$$
$$now' = now$$
$$wpm' = wpm$$
$$toneHz' = toneHz$$
$$iambicMode' = iambicMode$$
$$hapticEnabled' = hapticEnabled$$

When both paddles are held simultaneously, dit takes priority for the first element. This is a common convention; the subsequent alternation will then produce dah-dit-dah-dit...

12.2 Element Duration Elapsed

When an element has played for its full duration, transition to gap phase:

EndElement

$\Delta Keyer$

$keyerPhase = playing$
$(currentElement = ditElement \land$
$\quad now \geq elementStartTime + ditDuration(wpm)) \lor (currentElement = dahElement \land$
$\quad now \geq elementStartTime + dahDuration(wpm))$
$keyerPhase' = gap$
$gapStartTime' = now$
$elementStartTime' = 0$
$lastPlayedElement' = currentElement$
$currentElement' = currentElement$
$pendingOpposite' = pendingOpposite$
$toneActive' = zfalse$
$hapticPending' = zfalse$
$ditHeld' = ditHeld$
$dahHeld' = dahHeld$
$hapticPaddle' = hapticPaddle$
$now' = now$
$wpm' = wpm$
$toneHz' = toneHz$
$iambicMode' = iambicMode$
$hapticEnabled' = hapticEnabled$

The element completes and the keyer enters a silent gap before potentially starting the next element.

12.3 Gap Elapsed with Next Element

When the inter-element gap completes and there's a next element to play:

EndGapWithNext

$\Delta Keyer$

$nextElement : Element$

$keyerPhase = gap$

$now \geq gapStartTime + gapDuration(wpm)$

$(pendingOpposite = ztrue \land$

$lastPlayedElement = ditElement \land nextElement = dahElement) \lor (pendingOpposite = ztrue \land$

$lastPlayedElement = dahElement \land nextElement = ditElement) \lor (pendingOpposite = zfalse \land ditHeld = zt$

$lastPlayedElement = ditElement \land nextElement = dahElement) \lor (pendingOpposite = zfalse \land ditHeld = zt$

$lastPlayedElement = dahElement \land nextElement = ditElement) \lor (pendingOpposite = zfalse \land ditHeld = zt$

$nextElement = ditElement) \lor (pendingOpposite = zfalse \land ditHeld = zfalse \land dahHeld = ztrue \land$

$nextElement = dahElement)$

$keyerPhase' = playing$

$currentElement' = nextElement$

$elementStartTime' = now$

$gapStartTime' = 0$

$pendingOpposite' = zfalse$

$lastPlayedElement' = lastPlayedElement$

$toneActive' = ztrue$

$(hapticEnabled = ztrue \Rightarrow hapticPending' = ztrue)$

$(hapticEnabled = zfalse \Rightarrow hapticPending' = zfalse)$

$(nextElement = ditElement \Rightarrow hapticPaddle' = dit)$

$(nextElement = dahElement \Rightarrow hapticPaddle' = dah)$

$ditHeld' = ditHeld$

$dahHeld' = dahHeld$

$now' = now$

$wpm' = wpm$

$toneHz' = toneHz$

$iambicMode' = iambicMode$

$hapticEnabled' = hapticEnabled$

The next element decision follows this priority:

1. If *pendingOpposite* is set, play the opposite of what just finished
2. If both paddles held (squeeze), alternate from what just finished
3. If only dit held, play dit
4. If only dah held, play dah
5. If nothing held, return to idle (see next schema)

12.4 Gap Elapsed to Idle

When the gap completes but no paddle is held:

EndGapToIdle

$\Delta Keyer$

$keyerPhase = gap$
$now \geq gapStartTime + gapDuration(wpm)$
$pendingOpposite = zfalse$
$ditHeld = zfalse$
$dahHeld = zfalse$
$keyerPhase' = keyerIdle$
$elementStartTime' = 0$
$gapStartTime' = 0$
$pendingOpposite' = zfalse$
$lastPlayedElement' = lastPlayedElement$
$currentElement' = currentElement$
$toneActive' = zfalse$
$hapticPending' = zfalse$
$ditHeld' = ditHeld$
$dahHeld' = dahHeld$
$hapticPaddle' = hapticPaddle$
$now' = now$
$wpm' = wpm$
$toneHz' = toneHz$
$iambicMode' = iambicMode$
$hapticEnabled' = hapticEnabled$

The keyer returns to idle, ready for the next paddle touch.

13 Timer Operations

13.1 Advance Time

The timer tick advances the clock:

AdvanceTime

$\Delta Keyer$
$delta? : Time$

$delta? \geq 1$
$now' = now + delta?$
$now + delta? \leq KEYER\_MODEL\_BOUND$
$ditHeld' = ditHeld$
$dahHeld' = dahHeld$
$keyerPhase' = keyerPhase$
$currentElement' = currentElement$
$elementStartTime' = elementStartTime$
$gapStartTime' = gapStartTime$
$pendingOpposite' = pendingOpposite$
$lastPlayedElement' = lastPlayedElement$
$toneActive' = toneActive$
$hapticPending' = hapticPending$
$hapticPaddle' = hapticPaddle$
$wpm' = wpm$
$toneHz' = toneHz$
$iambicMode' = iambicMode$
$hapticEnabled' = hapticEnabled$

The implementation calls this at 1ms intervals using CADisplayLink for smooth, jitter-free timing.

13.2 Tick

The complete timer callback checks for state transitions after advancing time. This is a promoted operation combining time advance with transition checks:

$$Tick \mathrel{\widehat=} AdvanceTime \fatsemi TickTransitions$$

Where *TickTransitions* is defined as:

TickTransitions

$$\Delta Keyer$$

$$(keyerPhase = keyerIdle \land (ditHeld = ztrue \lor dahHeld = ztrue)$$
$$\land \Xi KeyerConfig \land \exists \Delta Keyer \bullet StartFromIdle) \lor (keyerPhase = playing \land$$
$$((currentElement = ditElement \land$$
$$now \geq elementStartTime + ditDuration(wpm)) \lor$$
$$(currentElement = dahElement \land$$
$$now \geq elementStartTime + dahDuration(wpm)))$$
$$\land \Xi KeyerConfig \land \exists \Delta Keyer \bullet EndElement) \lor (keyerPhase = gap \land now \geq gapStartTime + gapDuration(w$$
$$\Xi KeyerConfig \land$$
$$((\exists \Delta Keyer; nextElement : Element \bullet EndGapWithNext) \lor$$
$$(\exists \Delta Keyer \bullet EndGapToIdle))) \lor (\Xi Keyer)$$

14 Configuration Operations

SetWpm

$$\Delta Keyer$$
$$newWpm? : \mathbb{N}$$

$$newWpm? \geq minWpm$$
$$newWpm? \leq maxWpm$$
$$wpm' = newWpm?$$
$$toneHz' = toneHz$$
$$iambicMode' = iambicMode$$
$$hapticEnabled' = hapticEnabled$$
$$ditHeld' = ditHeld$$
$$dahHeld' = dahHeld$$
$$keyerPhase' = keyerPhase$$
$$currentElement' = currentElement$$
$$elementStartTime' = elementStartTime$$
$$gapStartTime' = gapStartTime$$
$$pendingOpposite' = pendingOpposite$$
$$lastPlayedElement' = lastPlayedElement$$
$$toneActive' = toneActive$$
$$hapticPending' = hapticPending$$
$$hapticPaddle' = hapticPaddle$$
$$now' = now$$

WPM changes take effect immediately for the next element. An element currently playing completes at its original timing.

SetToneHz

$\Delta Keyer$
$newToneHz? : \mathbb{N}$

$newToneHz? \geq minToneHz$
$newToneHz? \leq maxToneHz$
$toneHz' = newToneHz?$
$wpm' = wpm$
$iambicMode' = iambicMode$
$hapticEnabled' = hapticEnabled$
$\Xi PaddleState$
$\Xi KeyerState$
$\Xi KeyerOutput$
$now' = now$

SetIambicMode

$\Delta Keyer$
$newMode? : IambicMode$

$iambicMode' = newMode?$
$wpm' = wpm$
$toneHz' = toneHz$
$hapticEnabled' = hapticEnabled$
$\Xi PaddleState$
$\Xi KeyerState$
$\Xi KeyerOutput$
$now' = now$

ToggleHaptic

$\Delta Keyer$

$(hapticEnabled = ztrue \land hapticEnabled' = zfalse) \lor (hapticEnabled = zfalse \land hapticEnabled' = ztrue)$
$wpm' = wpm$
$toneHz' = toneHz$
$iambicMode' = iambicMode$
$\Xi PaddleState$
$\Xi KeyerState$
$\Xi KeyerOutput$
$now' = now$

15 Query Operations

GetTimings

$\Xi Keyer$
$ditMs! : Time$
$dahMs! : Time$
$gapMs! : Time$

$ditMs! = ditDuration(wpm)$
$dahMs! = dahDuration(wpm)$
$gapMs! = gapDuration(wpm)$

IsPlaying

$\Xi Keyer$
$playing! : ZBOOL$

$(keyerPhase = playing \land playing! = ztrue) \lor (keyerPhase \neq playing \land playing! = zfalse)$

16 Iambic Mode A Variant

Mode A differs from Mode B in one key respect: when both paddles are released during an element, the element is truncated immediately rather than completing.

TruncateElement

$$\Delta Keyer$$

$$keyerPhase = playing$$
$$iambicMode = modeA$$
$$ditHeld = zfalse$$
$$dahHeld = zfalse$$
$$keyerPhase' = keyerIdle$$
$$elementStartTime' = 0$$
$$gapStartTime' = 0$$
$$pendingOpposite' = zfalse$$
$$currentElement' = currentElement$$
$$lastPlayedElement' = currentElement$$
$$toneActive' = zfalse$$
$$hapticPending' = zfalse$$
$$ditHeld' = ditHeld$$
$$dahHeld' = dahHeld$$
$$hapticPaddle' = hapticPaddle$$
$$now' = now$$
$$wpm' = wpm$$
$$toneHz' = toneHz$$
$$iambicMode' = iambicMode$$
$$hapticEnabled' = hapticEnabled$$

For Mode A, the tick transitions would include this additional case. Most operators find Mode A produces choppier-sounding code and prefer Mode B.

17 Error Schemas

For totalized operations:

PaddleAlreadyHeld

$$\Xi Keyer$$
$$p? : Paddle$$

$$(p? = dit \land ditHeld = ztrue) \lor (p? = dah \land dahHeld = ztrue)$$

PaddleNotHeld

$$\Xi Keyer$$
$$p? : Paddle$$

$$(p? = dit \land ditHeld = zfalse) \lor (p? = dah \land dahHeld = zfalse)$$

InvalidWpm

$$\Xi Keyer$$
$$newWpm? : \mathbb{N}$$

$$newWpm? < minWpm \lor newWpm? > maxWpm$$

InvalidToneHz

$$\Xi Keyer$$
$$newToneHz? : \mathbb{N}$$

$$newToneHz? < minToneHz \lor newToneHz? > maxToneHz$$

18 Total Operations

$PaddleDownTotal \mathrel{\widehat{=}} PaddleDown \lor PaddleAlreadyHeld$

$PaddleUpTotal \mathrel{\widehat{=}} PaddleUp \lor PaddleNotHeld$

$SetWpmTotal \mathrel{\widehat{=}} SetWpm \lor InvalidWpm$

$SetToneHzTotal \mathrel{\widehat{=}} SetToneHz \lor InvalidToneHz$

19 Integration with Training Session

The keyer integrates with the Koch Trainer's *TrainingSession* through these connections:

19.1 Radio Mode Constraint

The keyer may only generate tones when the radio is in transmit mode. This models the half-duplex constraint from koch_trainer.tex:

KeyerRadioConstraint

$Keyer$
$radioMode : RadioMode$

$toneActive = ztrue \Rightarrow radioMode = transmitting$
$radioMode = off \Rightarrow keyerPhase = keyerIdle$

When the radio is off or receiving, the keyer must be idle.

19.2 Tone Synchronization

The keyer's *toneActive* output drives the *TrainingSession toneActive* field. The operation sequences are:

1. User begins touch $\rightarrow$ *PaddleDown* $\rightarrow$ *StartFromIdle* $\rightarrow$ $toneActive := ztrue$ $\rightarrow$ *ActivateTone* (in TrainingSession)
2. Element completes $\rightarrow$ *EndElement* $\rightarrow$ $toneActive := zfalse$ $\rightarrow$ *DeactivateTone* (in TrainingSession)

19.3 Session Flow

During send training:

1. *InitSession* sets $direction := send$
2. *CompleteIntroduction* sets $radioMode := receiving$
3. User presses paddle $\rightarrow$ *EnterTransmitMode* sets $radioMode := transmitting$
4. Keyer generates elements via *StartFromIdle*, *EndElement*, *EndGapWithNext*
5. User finishes character $\rightarrow$ timeout $\rightarrow$ *ExitTransmitMode* sets $radioMode := receiving$
6. System compares keyed pattern to expected $\rightarrow$ *RecordCorrectResponse* or *RecordIncorrectResponse*
7. Repeat until proficiency met

20 System Invariants

The following properties hold for any reachable keyer state:

1. **Tone-phase consistency**: $toneActive = ztrue \Leftrightarrow keyerPhase = playing$
2. **Timing validity**: When $keyerPhase = playing$, $elementStartTime \leq now$
3. **Gap validity**: When $keyerPhase = gap$, $gapStartTime \leq now$
4. **Idle silence**: When $keyerPhase = keyerIdle$, $toneActive = zfalse$
5. **Duration bounds**: For any $w \in minWpm \,..\, maxWpm$, $ditDuration(w) \geq 30 \land ditDuration(w) \leq 240$

21 Precondition Summary

Operation	Precondition
PaddleDown	$p? = dit \Rightarrow ditHeld = zfalse$; $p? = dah \Rightarrow dahHeld = zfalse$
PaddleUp	$p? = dit \Rightarrow ditHeld = ztrue$; $p? = dah \Rightarrow dahHeld = ztrue$
StartFromIdle	$keyerPhase = keyerIdle \land (ditHeld = ztrue \lor dahHeld = ztrue)$
EndElement	$keyerPhase = playing \land now \geq elementStartTime + duration$
EndGapWithNext	$keyerPhase = gap \land now \geq gapStartTime + gapDuration(wpm) \land (pendingOpposite = ztrue \lor ditHeld = ztrue \lor dahHeld = ztrue)$
EndGapToIdle	$keyerPhase = gap \land now \geq gapStartTime + gapDuration(wpm) \land pendingOpposite = zfalse \land ditHeld = zfalse \land dahHeld = zfalse$
SetWpm	$minWpm \leq newWpm? \leq maxWpm$
SetToneHz	$minToneHz \leq newToneHz? \leq maxToneHz$
TruncateElement	$keyerPhase = playing \land iambicMode = modeA \land ditHeld = zfalse \land dahHeld = zfalse$

22 Validation

This specification can be validated with:

- **fuzz**: Type-checking the combined specification
- **probcli -init**: Verifying initialization succeeds
- **probcli -cbc_deadlock**: Checking no deadlock states exist
- **probcli -cbc_assertions**: Verifying invariants hold

The *KEYER_MODEL_BOUND* constant enables finite model checking. For combined validation with koch_trainer.tex, ensure both specifications use compatible bounds and the shared types (*ZBOOL*, *RadioMode*) are defined consistently.

23 Implementation Notes

24 Process and Threading Design

The Z specification models state transitions but does not address *when* or *where* these transitions execute. This section defines the process architecture required for correct implementation.

Key principle: Z models *what* state changes occur. This section models *how* those changes are triggered and synchronized across threads.

24.1 Thread Model

The implementation involves four distinct execution contexts:

Context	Timing	Responsibility
UIKit Events	Event-driven (instant)	Touch began/ended delivery
Main Thread	60-120 FPS (8-16ms)	ViewModel state, SwiftUI rendering
Display Link	60-120 FPS (8-16ms)	Keyer tick loop, element timing
Audio RT	44.1kHz (23μs/sample)	Tone generation, waveform output

24.2 Component Responsibilities

24.2.1 PaddleInputManager (Main Thread)

Receives touch events from UIKit and manages paddle memory:

- **Input:** `touchesBegan`, `touchesEnded` from UIKit
- **State:** `ditHeld`, `dahHeld`, `ditMemory`, `dahMemory`
- **Output:** Callback to keyer engine

Paddle memory latches press events to survive timing gaps between touch delivery and keyer tick:

```
func handleTouchBegan(paddle: Paddle) {
    if paddle == .dit {
        ditHeld = true
        ditMemory = true   // Latch until serviced
    }
    // ... similar for dah
    onPaddleChanged(PaddleState(...))
}
```

24.2.2 IambicKeyerEngine (Main Thread via CADisplayLink)

Implements the Z state machine with precise timing:

- **Timing source:** CADisplayLink at 120 FPS preferred
- **Input:** `PaddleState` from input manager
- **State:** Implements `KeyerState` schema
- **Output:** Callbacks for element start/end, pattern completion

The keyer reads paddle memory and clears it when serviced:

```
private func selectNextElement() -> Element {
    if ditMemory {
        ditMemory = false  // Clear after servicing
        return .dit
    }
    if dahMemory {
        dahMemory = false
        return .dah
    }
    // Fall back to held state
    if paddle.ditHeld { return .dit }
    if paddle.dahHeld { return .dah }
}
```

24.2.3 PatternTracker (Main Thread)

Single owner of character boundary detection:

- **Input:** Element completed callbacks from keyer
- **State:** Current pattern string, idle timer
- **Output:** Pattern completion callback to ViewModel

Critical: Only one component owns character timeout. The failed implementation had competing timeouts in both keyer and ViewModel.

24.2.4 AudioController (Main → Audio RT)

Bridges main thread commands to real-time audio:

- **Input:** `toneActive` changes from keyer
- **State:** Lock-protected flags readable by audio callback
- **Output:** Waveform samples to AVAudioEngine

24.3 Timing Diagram

Successful element playback sequence:

```
TIME    CONTEXT         EVENT                          STATE
-------------------------------------------------------------------------
T+0ms   UIKit           touchesBegan                   ditHeld=true
T+0ms   Main            onPaddleChanged callback       ditMemory=true
T+0ms   Main            keyer checks (if idle)         phase=playing
T+0ms   Main            onToneStart callback           -
T+0ms   Main->Audio     activateTone()                 isToneActive=true
T+1ms   Audio RT        next buffer renders tone       speaker output

T+8ms   DisplayLink     tick: elapsed=8ms < 92ms       (no change)
T+16ms  DisplayLink     tick: elapsed=16ms < 92ms      (no change)
...
T+92ms  DisplayLink     tick: elapsed >= 92ms          phase=gap
T+92ms  Main            onToneStop callback            -
T+92ms  Main->Audio     deactivateTone()               isToneActive=false
```

24.4 Thread Safety Patterns

24.4.1 Main Thread (@MainActor)

All ViewModel state must be accessed on the main thread:

```
@MainActor
final class SendTrainingViewModel: ObservableObject {
    @Published var currentPattern: String = ""

    // Callbacks from keyer run on main thread
    func handlePatternComplete(_ pattern: String) {
        currentPattern = pattern  // Safe: @MainActor
    }
}
```

24.4.2 Audio Thread (NSLock)

Audio callbacks cannot use Swift actors. Use explicit locking:

```
final class ToneGenerator: @unchecked Sendable {
    private let lock = NSLock()
    private var _isToneActive = false

    var isToneActive: Bool {
        lock.lock()
        defer { lock.unlock() }
        return _isToneActive
    }
}
```

24.4.3 Callback Threading

All keyer callbacks should be explicitly `@MainActor`:

```
init(
    onToneStart: @escaping @MainActor (Double) -> Void,
    onToneStop: @escaping @MainActor () -> Void,
    onPatternComplete: @escaping @MainActor (String) -> Void
)
```

24.5 Event-Driven vs Polling

Anti-pattern (from failed implementation):

```
// WRONG: Polling reads stale state
func updatePaddle(...) {
    keyer.queueElement(.dit)
    currentPattern = keyer.currentPattern  // Stale!
}
```

Correct pattern:

```
// RIGHT: Event-driven updates
keyer.onPatternUpdated = { pattern in
    Task { @MainActor in
        self.currentPattern = pattern
    }
}
```

24.6 Single Timeout Owner

The failed implementation had two competing timeout mechanisms:

1. Keyer's idle timeout (415ms at 13 WPM)
2. ViewModel's input timer (variable duration)

Solution: Choose one owner. Two valid designs:
Option A: Keyer owns character boundaries

- Keyer detects idle timeout, emits `onPatternComplete`
- ViewModel receives completed patterns, no timer needed
- Keyer controls all timing

Option B: ViewModel owns character boundaries

- Keyer emits `onElementCompleted` for each element
- ViewModel accumulates pattern, manages single timeout
- ViewModel decides when pattern is complete

24.7 Latency Budget

End-to-end latency requirements:

Segment	Budget
Touch to keyer state change	$<$ 1ms
Keyer to audio flag change	$<$ 1ms
Audio flag to speaker output	$<$ 3ms (next buffer)
Total touch-to-audio	$<$ **5ms**

At 25 WPM, a dit is 48ms. Latency under 10% of element duration (5ms) is imperceptible. Latency above 20ms becomes noticeable as sluggishness.

25 Implementation Notes

25.1 Timer Resolution

Use CADisplayLink (iOS) at maximum frame rate (120 FPS on ProMotion devices). This provides 8ms resolution, sufficient for 13 WPM (92ms dit). For higher speeds (25+ WPM), consider a dedicated `DispatchSourceTimer` at 1ms intervals.

At 25 WPM, a dit is only 48ms—timing jitter above 5ms becomes audible.

25.2 Paddle Memory

Implement dit/dah memory per the Curtis 8044 patent. When the opposite paddle is pressed during an element, latch that press until the keyer services it. This prevents lost input when paddle is released before the next tick.

```
// Set memory on press (edge-triggered)
if input.ditPressed && !previousInput.ditPressed {
    ditMemory = true
}
// Clear memory when element starts
private func startElement(_ element: Element) {
    if element == .dit { ditMemory = false }
    if element == .dah { dahMemory = false }
    // ...
}
```

25.3 Haptic Latency

iOS Core Haptics has approximately 10-15ms latency. Pre-warm the haptic engine on view appearance to avoid additional latency on first touch.

25.4 Audio Engine Configuration

Use continuous audio session mode:

- Start `AVAudioEngine` at session begin, run until session end
- Control tone via `isToneActive` flag, not engine start/stop
- Engine start/stop adds 50-100ms latency; flag toggle is sample-accurate

Total touch-to-audio latency should be under 5ms for comfortable operation up to 25 WPM.

25.5 Lessons from Failed Implementation

The first implementation (PR #68) failed due to:

1. **Missing paddle memory:** Quick taps lost between ticks
2. **Competing timeouts:** Keyer and ViewModel both had timers
3. **Polling state sync:** ViewModel read stale keyer state
4. **idleStartTime bug:** Not reset on re-entry to playing phase

See `docs/keyer-post-mortem.md` for detailed analysis.
See `docs/threading.md` for comprehensive Swift concurrency patterns.