1ST MAKER SPACE
Design.
Build.
Sustain.

Frog Microcontroller Trainer (FMT): Chess Timer

Length of Time: 2 days x 55 Minutes

Recommended for Developmental

Contents

- Overview
- Objectives
- Standards
- Materials
- Lesson Elements
- Resources

Overview:

The lesson centers on a chess timer program built on the 1ST Maker Frog board that simulates a real chess clock using embedded programming concepts.

At its core, the program is a finite state machine that manages different game states (setup, player A running, player B running, paused, and game over). It tracks each player's remaining time using a subtract-elapsed timing method, switches turns based on button release events, and supports features like pause/resume gestures and optional Fischer time increments.

Objectives:

Students will be able to:

•Explain how an enum provides named, type-safe constants for a finite state machine and why it is preferred over raw integer constants.
•Describe how a struct groups related variables into a single reusable unit, and trace how the ButtonTracker struct is used to manage two independent buttons with identical logic.
•Explain the subtract-elapsed timing technique used in tickActiveClock() and why resetting lastTickMs at every turn switch is essential to prevent time jumps.
•Trace the complete pause and resume gesture — including the hold timer, the armed/pending state, the release guard, and the event-clearing step — and explain why each piece is necessary.
•Describe the role of INCREMENT_SECONDS as a configurable constant and explain how the Fischer increment rule works in chess.
•Read and interpret M:SS formatted output produced by snprintf() and explain why snprintf() is used instead of print() calls.

Computer Science Teacher Association Standards:

- 3A-CS-01 – Explain how abstractions hide underlying implementation details of computing systems embedded in everyday objects.
- 3A-AP-13 – Create prototypes that use algorithms to solve computational problems based on personal interests.
- 3A-AP-15 – Justify the selection of specific control structures in terms of readability and performance.
- 3A-AP-17 – Decompose problems into smaller components through systematic analysis, using constructs such as procedures and modules.
- 3A-AP-21 – Evaluate and refine computational artifacts to make them more usable and accessible.

Materials:

- Frog Microcontroller Trainer from 1st Maker Space
- USB Power Cord
- Arduino IDE
- Chess Timer program loaded onto each Frog
- PC or Mac
- Chromebooks if using Arduino Create for Education App
- Copies of Student Handout

Preparation:

- Confirm the Frog boards, library, and Chess Timer sketch compile and upload successfully.
- Run the program yourself and press each button to see the behavior.
- Decide which extensions are appropriate for your students' level.
- Print or distribute the student handouts.

Background Information:

A chess clock gives each player a fixed pool of time for the entire game. The clock counts down only for the player whose turn it is. When a player finishes their move, they press their button: this stops their clock and starts their opponent's. If a player's time reaches zero, they lose on time regardless of the position on the board.
The chess timer is an ideal embedded programming project because it is a real, familiar device with well-understood rules, yet it requires precise non-blocking timing, multi-button gesture recognition, a clear state machine, and a formatted display — all skills that transfer directly to professional embedded work.

Named States:
Chess_Timer.ino uses a C enum to define the five clock states: STATE_SET, STATE_RUN_A, STATE_RUN_B, STATE_PAUSED, and STATE_OVER. An enum is a user-defined type whose values are named integer constants. The compiler assigns

sequential integers starting from zero, but the programmer (and the reader) uses the names, never the numbers.
Using an enum rather than raw constants (const uint8_t STATE_SET = 0, etc.) provides two benefits. First, the type of the variable state is ClockState, not uint8_t; the compiler can warn if a value of the wrong type is assigned. Second, the complete set of valid states is declared in one place, making it impossible to accidentally introduce an undeclared state. This is an example of using the type system to prevent bugs rather than relying on programmer discipline alone.

The Fischer Increment:
INCREMENT_SECONDS is currently set to zero, making this a simple sudden-death timer. Setting it to a positive value implements the Fischer increment rule: after a player presses their button to end their turn, that many seconds are added back to their remaining time before the clock switches. The intent is to guarantee each player at least some time per move even in a very fast game.
The increment is applied in handleStateMachine() immediately before switchTurnTo() is called. Students can verify this by finding the lines timeA_ms += (int32_t)INCREMENT_SECONDS * 1000 and timeB_ms += (int32_t)INCREMENT_SECONDS * 1000 and confirming they appear before the switch, not after.

Program Structure:

States and Transitions

State	Meaning	Transitions In	Transitions Out
STATE_SET	Startup: pot selects time preset, timers shown but not running.	Power-on / hardware reset.	Button A or B released → STATE_RUN_A or STATE_RUN_B.
STATE_RUN_A	Player A's clock counting down.	From STATE_SET (A released) or STATE_RUN_B (B released).	A released → STATE_RUN_B. Either timer hits zero → STATE_OVER. Both held → STATE_PAUSED.
STATE_RUN_B	Player B's clock counting down.	From STATE_SET (B released) or STATE_RUN_A (A released).	B released → STATE_RUN_A. Either timer hits zero → STATE_OVER. Both held → STATE_PAUSED.
STATE_PAUSED	Both clocks frozen. Awaiting resume gesture.	Both held ≥ PAUSE_HOLD_MS while running.	Both held ≥ PAUSE_HOLD_MS while paused (arm), then both released → prevRunState.
STATE_OVER	One player's time has reached zero. Game ended.	Either timeA_ms or timeB_ms ≤ 0.	Hardware reset button only.

The ButtonTracker Struct

Each ButtonTracker instance holds the complete debounce state for one button. The struct fields are:

Field	Type	Purpose
pin	uint8_t	The Arduino pin number to read with digitalRead().
stablePressed	bool	The current debounced (stable) state. True = button is held down.
lastStablePressed	bool	The stable state from the previous loop iteration, used to detect transitions.
lastChangeMs	uint32_t	Timestamp of the last time the raw reading matched the stable state. Used to measure the DEBOUNCE_MS hold-off.
pressedEvent	bool	Set true for exactly one loop iteration when the button transitions to pressed. Cleared at the start of each updateButtons() call.
releasedEvent	bool	Set true for exactly one loop iteration when the button transitions to released. Turn switches are triggered on release, not press.

Note that turn switches are intentionally triggered on button release rather than button press. This matches real chess clock behavior: the player's clock stops the moment they lift their finger, not the moment they press down. It also prevents a button held from one state accidentally triggering an action in the next.

Key Global Variables

Variable	Type	Purpose
state	ClockState	Current FSM state. The central dispatch variable.
prevRunState	ClockState	Remembers whether RUN_A or RUN_B was active before a pause so the correct player's clock resumes.
timeA_ms / timeB_ms	int32_t	Remaining time for each player in milliseconds. Signed so they can briefly go negative before being clamped to zero.
lastTickMs	uint32_t	Timestamp of the last tickActiveClock() call. dt = nowMs − lastTickMs gives the exact milliseconds to subtract.
bothPressStartMs	uint32_t	Timestamp when both buttons were first pressed simultaneously. Used to measure the hold duration for pause/resume.
bothGestureUsed	bool	Prevents the pause/resume action from firing more than once per physical hold.
resumePending	bool	Set true when the resume is armed. Clock only resumes after BOTH buttons are physically released.
ignoreReleasesUntilBothUp	bool	Suppresses release events immediately after a pause/resume hold to prevent the physical release from triggering a turn switch.

Lesson Elements	
Introduction -5 minutes	Show a physical chess clock (or a photo). Ask: "If you were going to build this yourself, what would be the hardest part to get right?" Elicit answers about timing accuracy, button handling, and pause behavior. Then show the Frog running Chess_Timer.ino and demonstrate a complete game including a pause.
Concept – 15 minutes	Introduce enum and struct. Draw the state diagram on the board. Explain the subtract-elapsed timing pattern— this is the conceptual core and deserves careful treatment. Ask students to predict what would happen if lastTickMs were not reset on a turn switch.
Code Walk– 15 minutes	Open Chess_Timer.ino. Walk through: (1) the ClockState enum and ButtonTracker struct declarations; (2) loop() and why updateButtons() must run before handleStateMachine(); (3) tickActiveClock() in detail; (4) the pause/resume block in handleStateMachine() step by step, naming each guard flag and explaining the failure mode it prevents.

Demo– 5minutes	Run the timer with two student volunteers. Narrate the state transitions aloud as they occur. Deliberately trigger the time-up condition to demonstrate the STATE_OVER behavior and the two-tone alert.
Hands-On – 15-20 Minutes	Students complete the worksheet. For classes with hardware access, also have students trigger each state transition deliberately — especially the hold-and-release pause/resume sequence.
Career Exploration: ● A good resource is the IDOE Career Explorer database. ● Another good resource for career information is the Bureau of Labor Statistics	
Practical Application: Chess tournaments, Debate timers, Game timers (board or digital games), Classroom timed activities, Competitive events with time limits	

Additional Resources:

- 1st Maker Space Frog Microcontroller Library
- 1st Maker Space Learn Arduino with the Frog Microcontroller

Performance Assessment/Check for Understanding :

- Was the student able to successfully load the sketch to the FMT?
- Check student handout for responses.

Additional Feedback: