Dragon Sensor
Design Problem Solution

Problem

- A digital **thermometer connected to A7..A0** reads the temperature inside the pressurized cabin of the SpaceX Dragon capsule (Above). The sensor samples at 1 second intervals.
- **If pressure in the cabin is lost, then A == 0**. (Space is very cold.)
- The software in the capsule uses a **lookup table "Temp_K_LUT"** to convert values of A into Kelvin temperatures. The values stored in the lookup table are 8-bit chars.
- The current temperature in the cabin (in Kelvin) needs to sent back to Earth, so ground control can monitor the current status of the capsule. **A temperature is sent back to earth by strobing B = 0xff for 10 ms, and then placing the temperature from the lookup table on B for 100 ms**.
- **If cabin pressure is lost (A==0), the system should alternately set B=0xff, then B=0x00**, repeatedly, at 100ms intervals, perhaps to warn other subsystems about the failure.

Solution #1
Single-Task

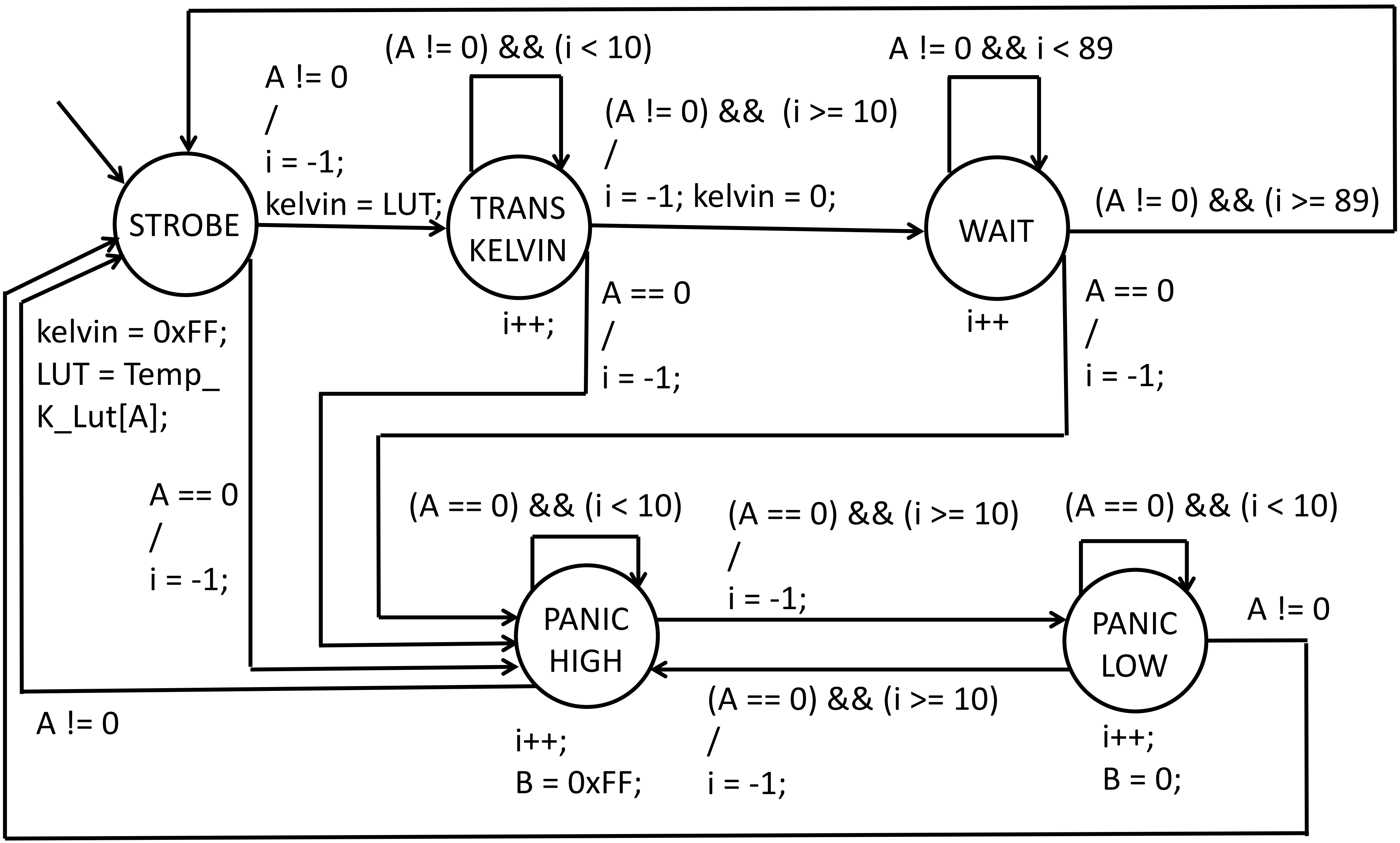
Overview

1. If (A != 0) // “Transmission” mode
   1.1 Strobe for 10ms; convert sensor A to Kelvin via LUT
   1.2 Transmit the kelvin value for 100ms
   1.3 Wait for the next 890ms to complete the 1s period

2. If (A == 0) // “PANIC” mode
   2.1 Alternate B=0xFF, B=0 for 100ms period

3. Swap between Transmission/PANIC mode
   3.1 Switch from Transmission to PANIC mode if A == 0
   3.2 Switch from PANIC to Transmission mode if A != 0

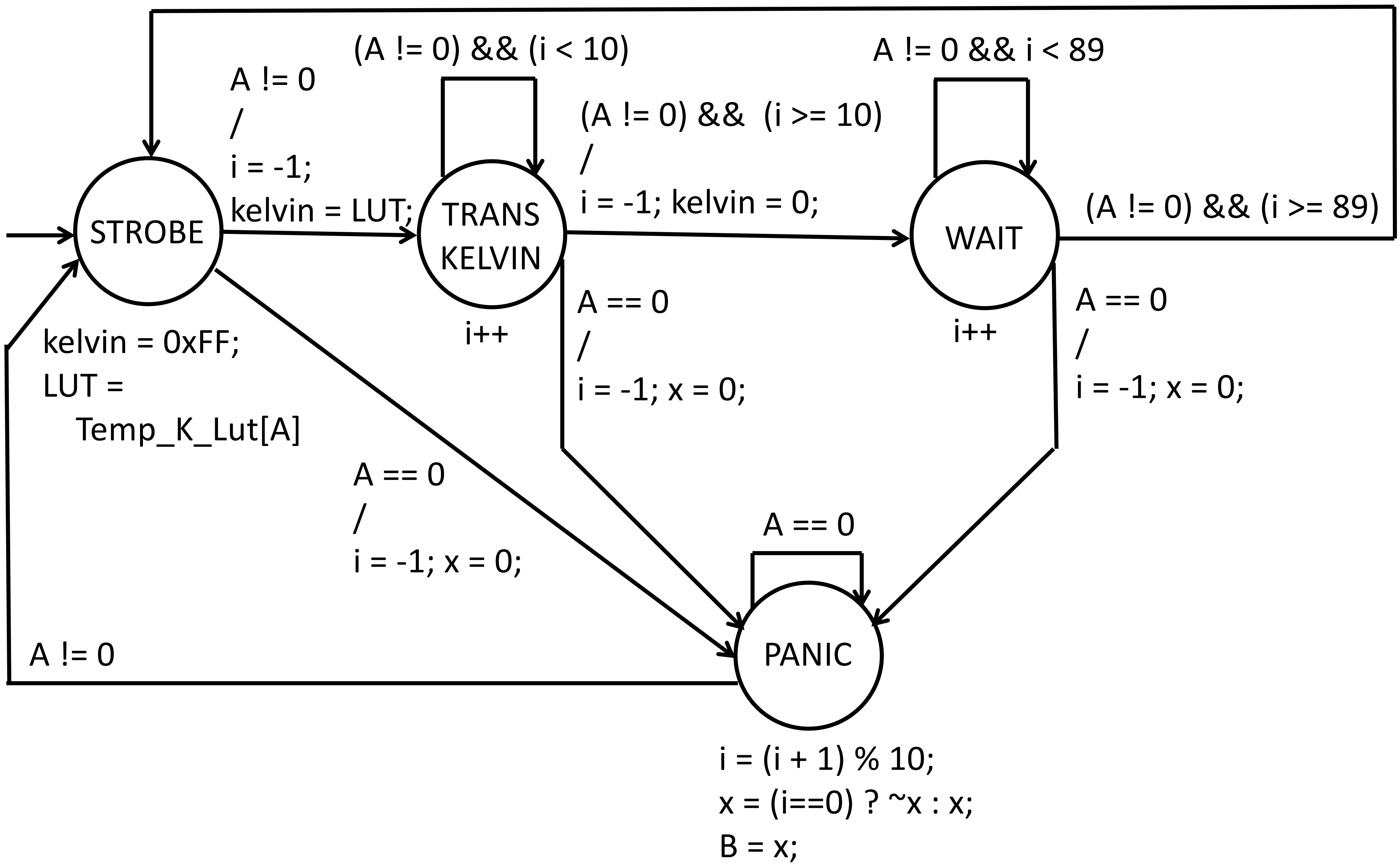
Solution #1, Single-Task



Solution #2
Single-Task (One “PANIC” mode state)

- Switching B from 0xFF to 0 and vice-versa is simply a bit inversion.
- Use a single state for “PANIC” mode
  - Use a variable x, initialized to zero
  - Every 100ms set x = ~x;
  - Output B = x
- Cleans up a bit of the mess

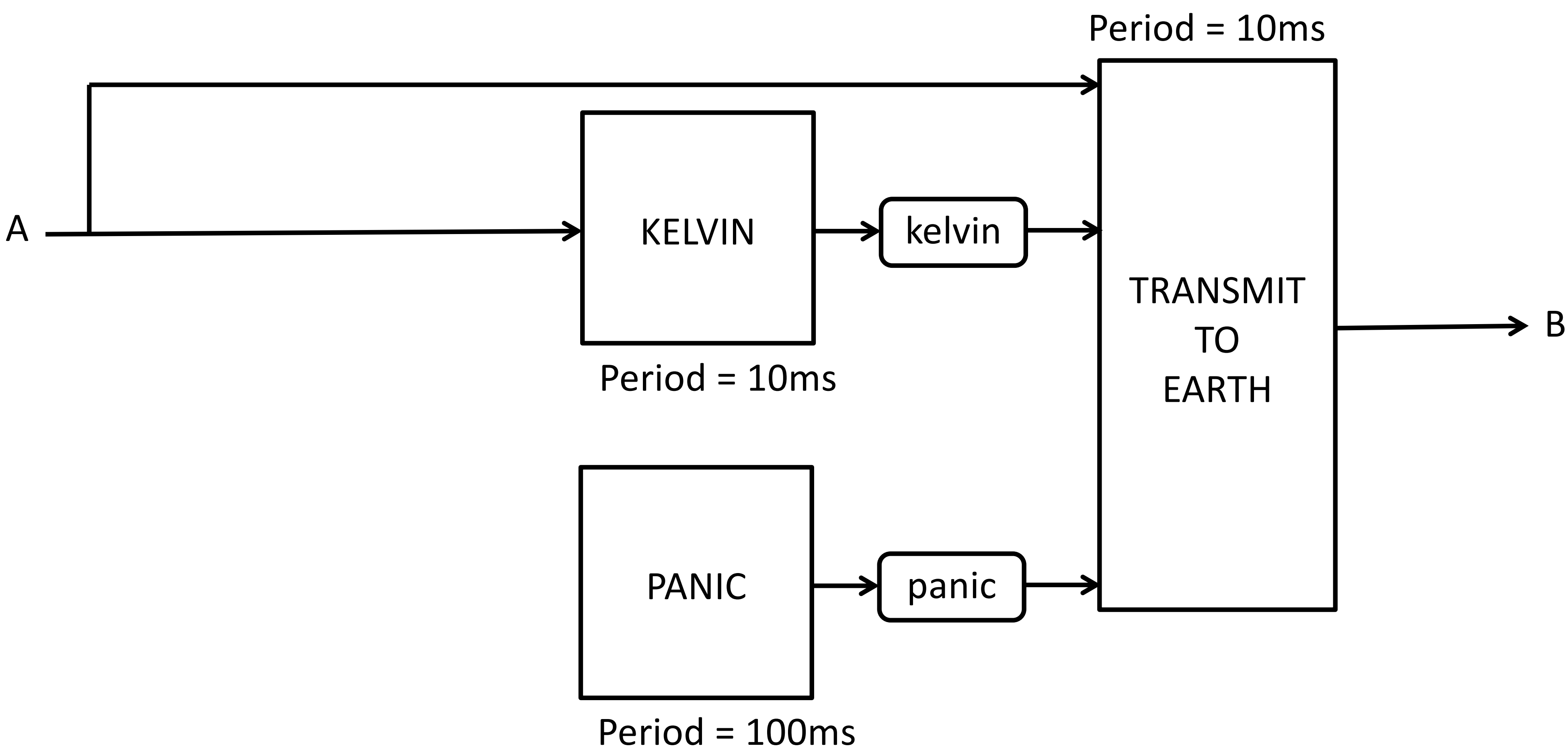
Solution #2, Single-Task



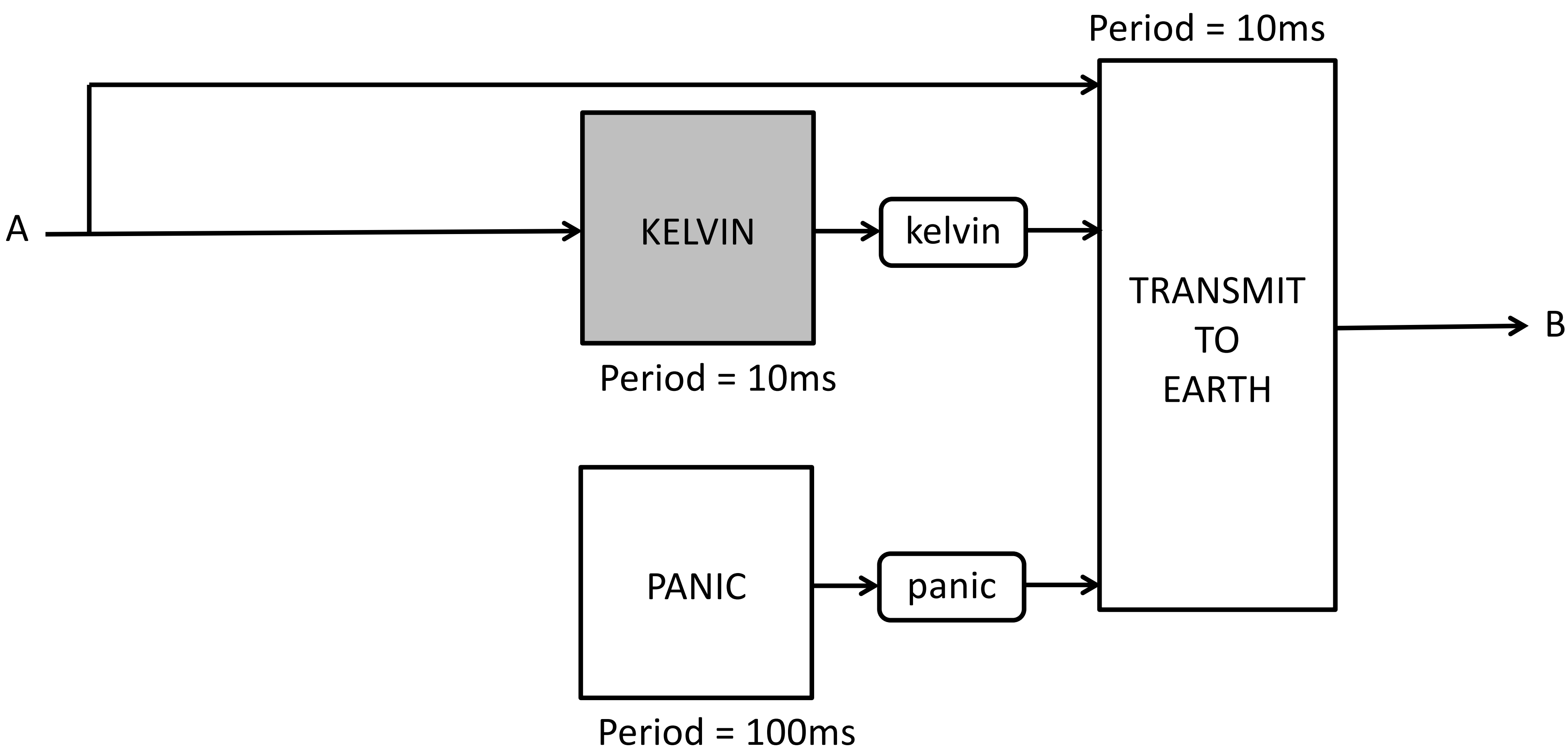
Solution #3
Concurrent SynchSM w/Shared Variables

- Separate “Transmission” and “PANIC” mode into separate tasks
- A third task acts as a multiplexer
  - If (A == 0) Output “PANIC” mode result
  - If (A != 0) Output “Transmission” mode result

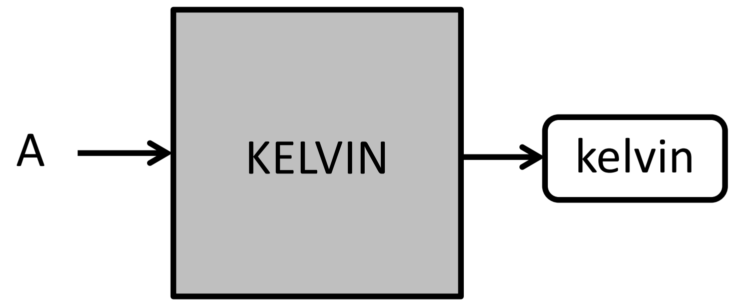
Solution #3
Concurrent SynchSM w/Shared Variables



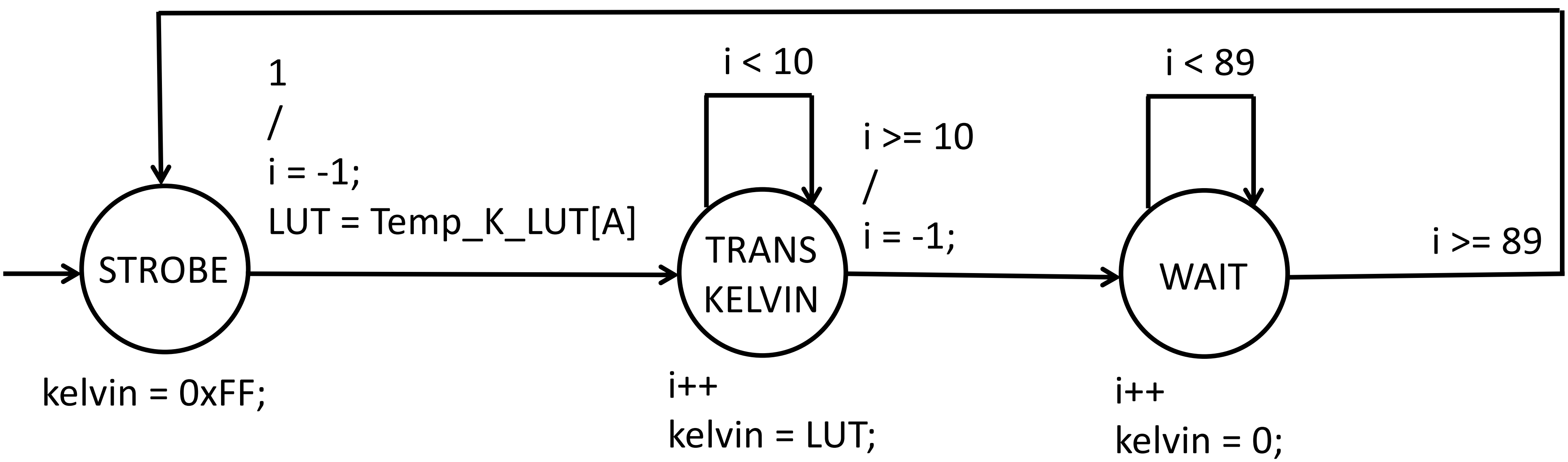
Solution #3
Concurrent SynchSM w/Shared Variables



KELVIN Task



Period = 10ms

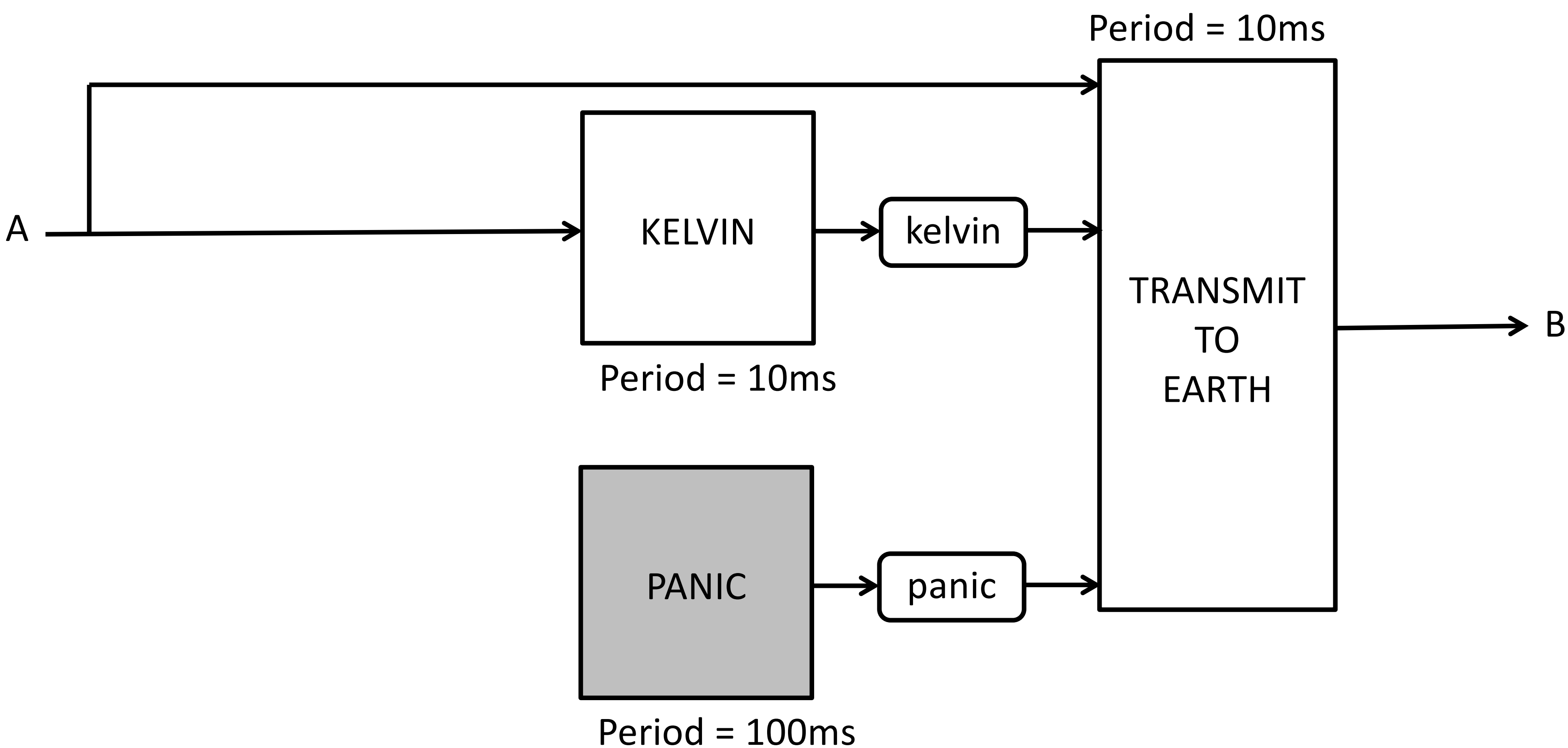


// Strobe for 10ms
//   and sample A

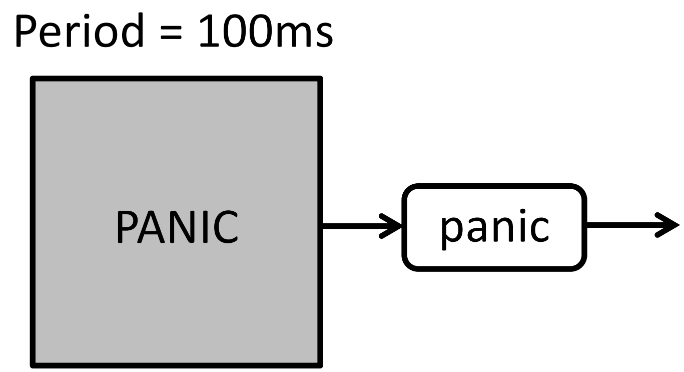
// Transmit for 100ms

// Wait for 890ms

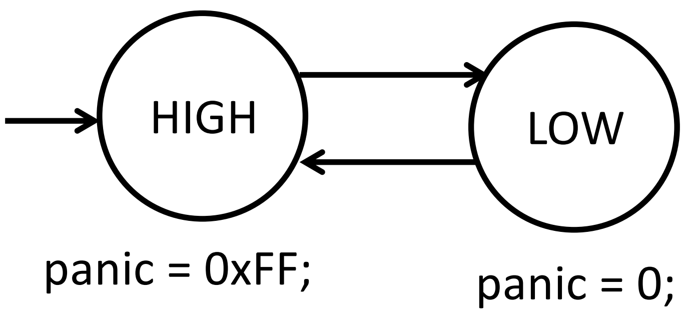
Solution #3
Concurrent SynchSM w/Shared Variables



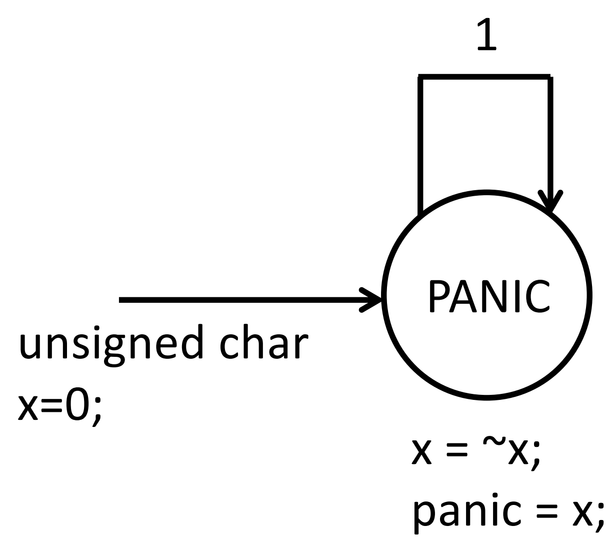
PANIC Task



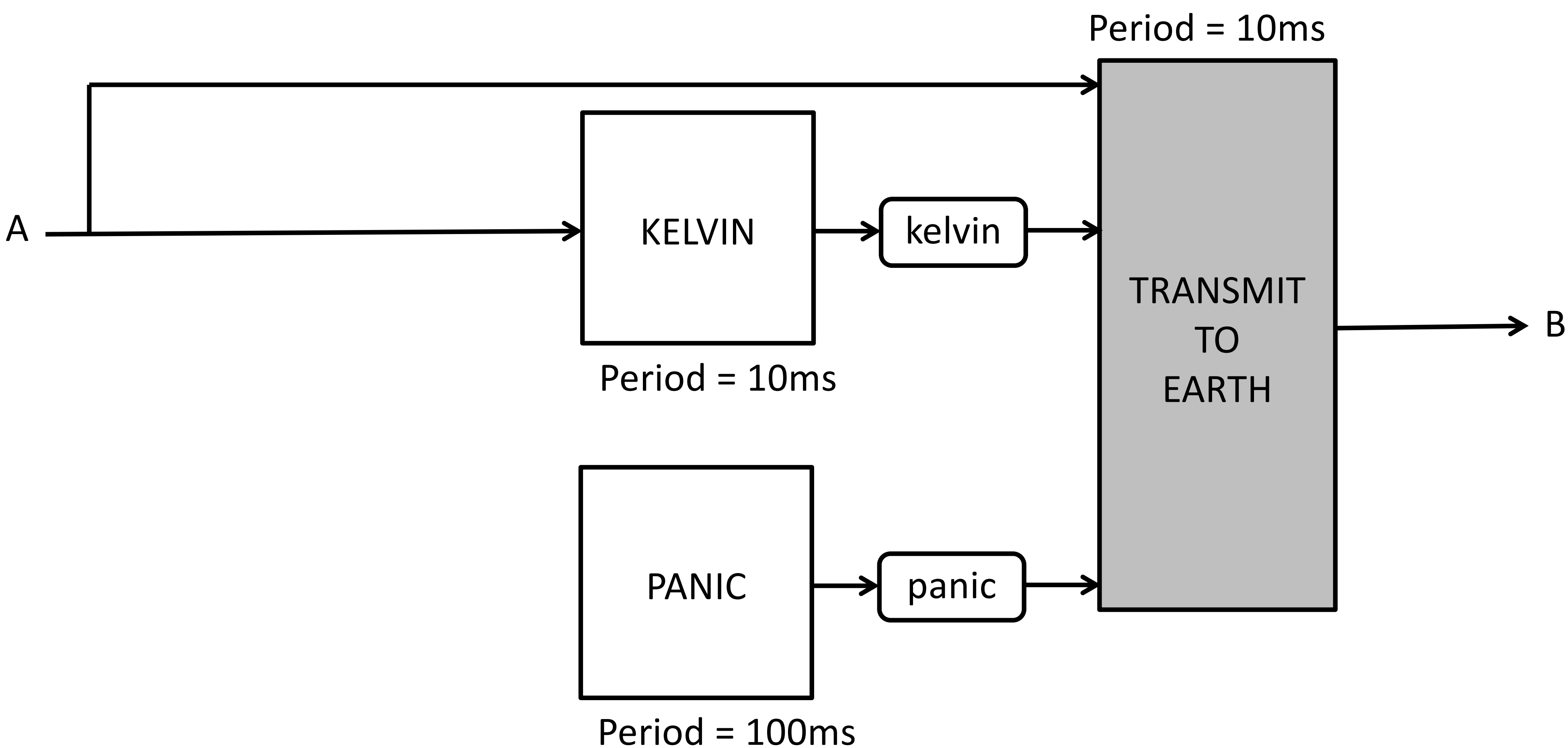
2-state Implementation



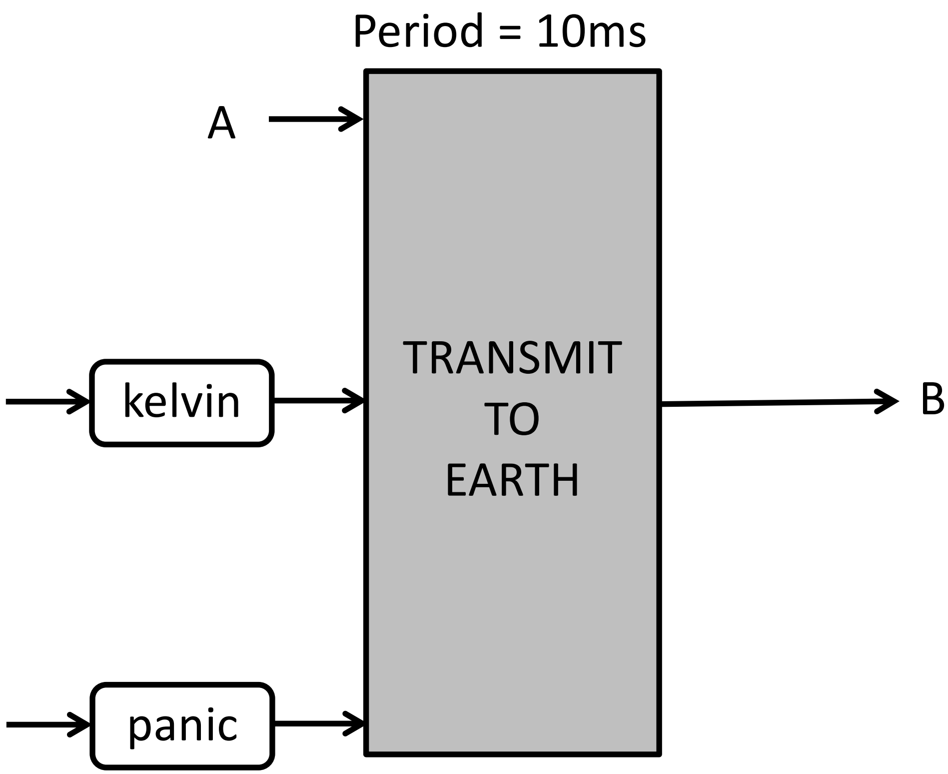
1-state Implementation

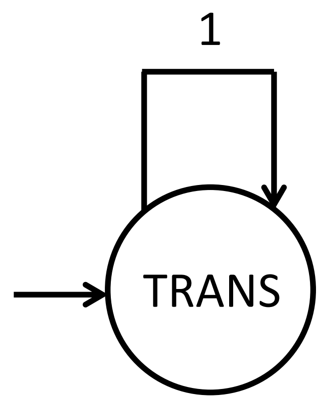


Solution #3
Concurrent SynchSM w/Shared Variables



TRANSMIT TO EARTH Task



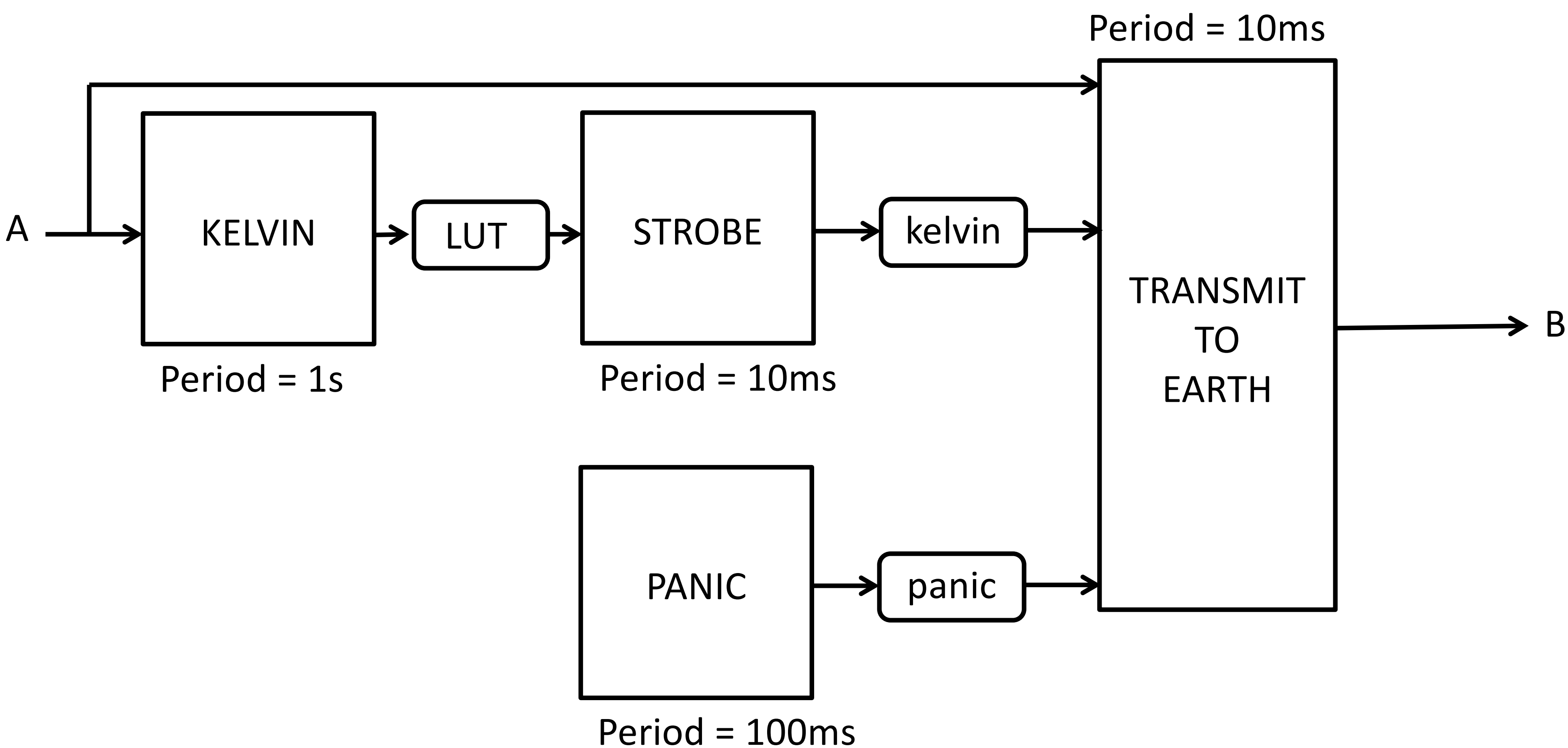


B = (A == 0) ? panic : kelvin;

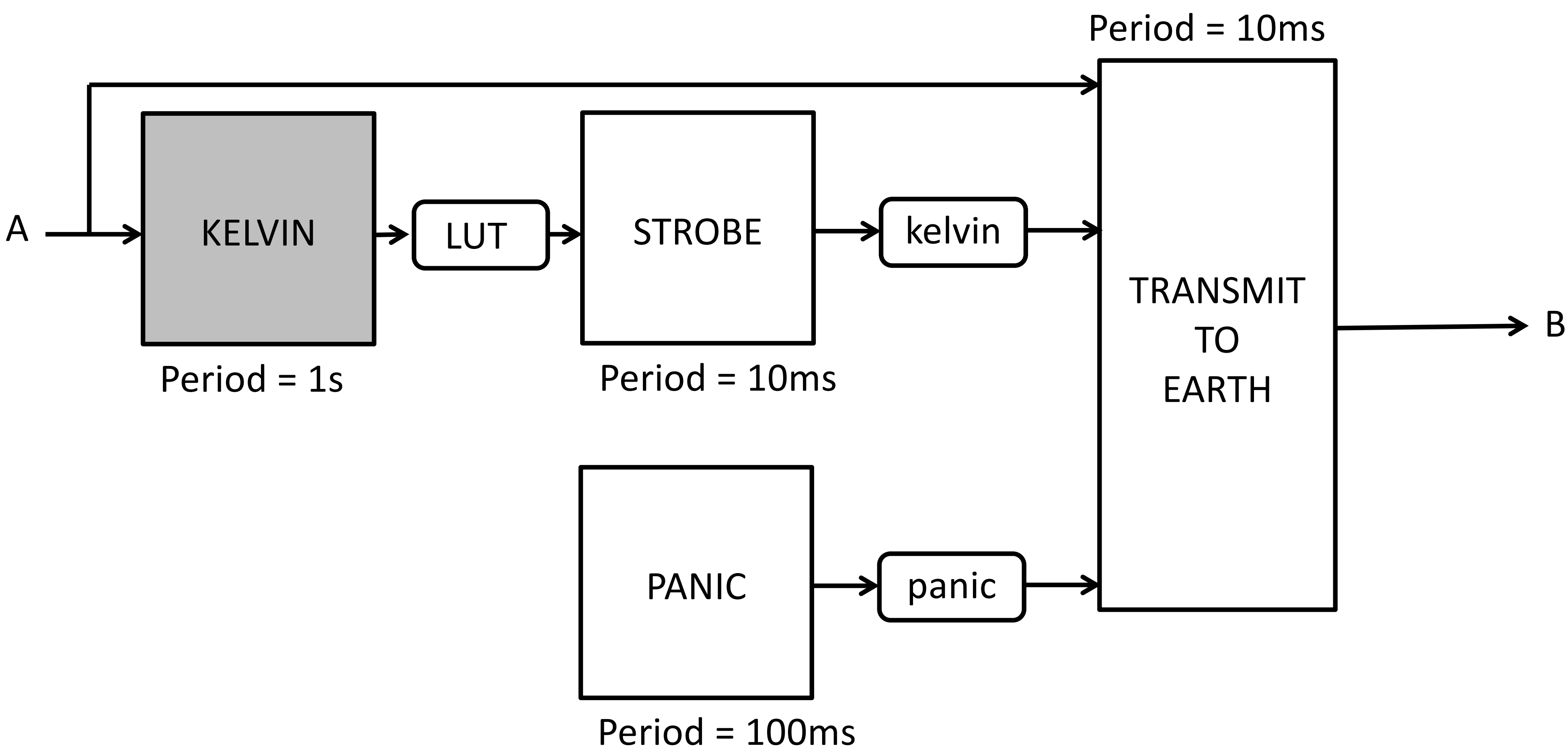
Solution #4
Concurrent synchSM w/Shared Variables

- Create a separate task with a 1 second period to perform the LUT-based conversion of A to kelvin
- Not much of an improvement over Solution #3, but leads to Solution #5 which is far more elegant

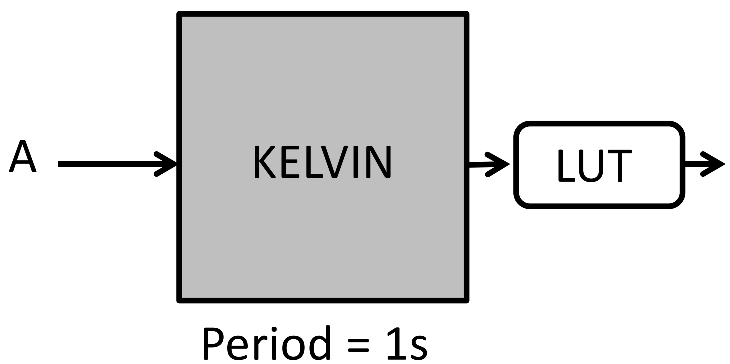
Solution #4
Concurrent SynchSM w/Shared Variables

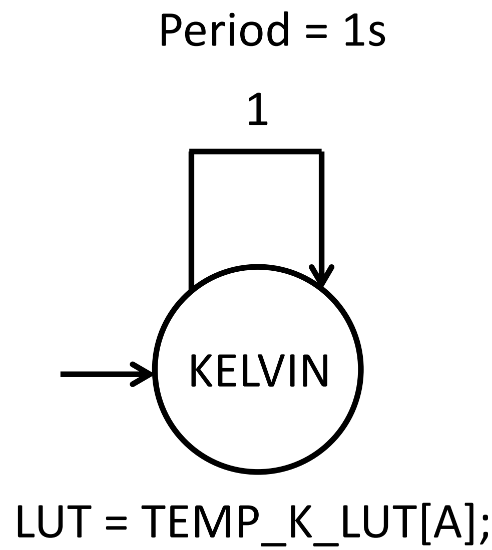


Solution #4
Concurrent SynchSM w/Shared Variables

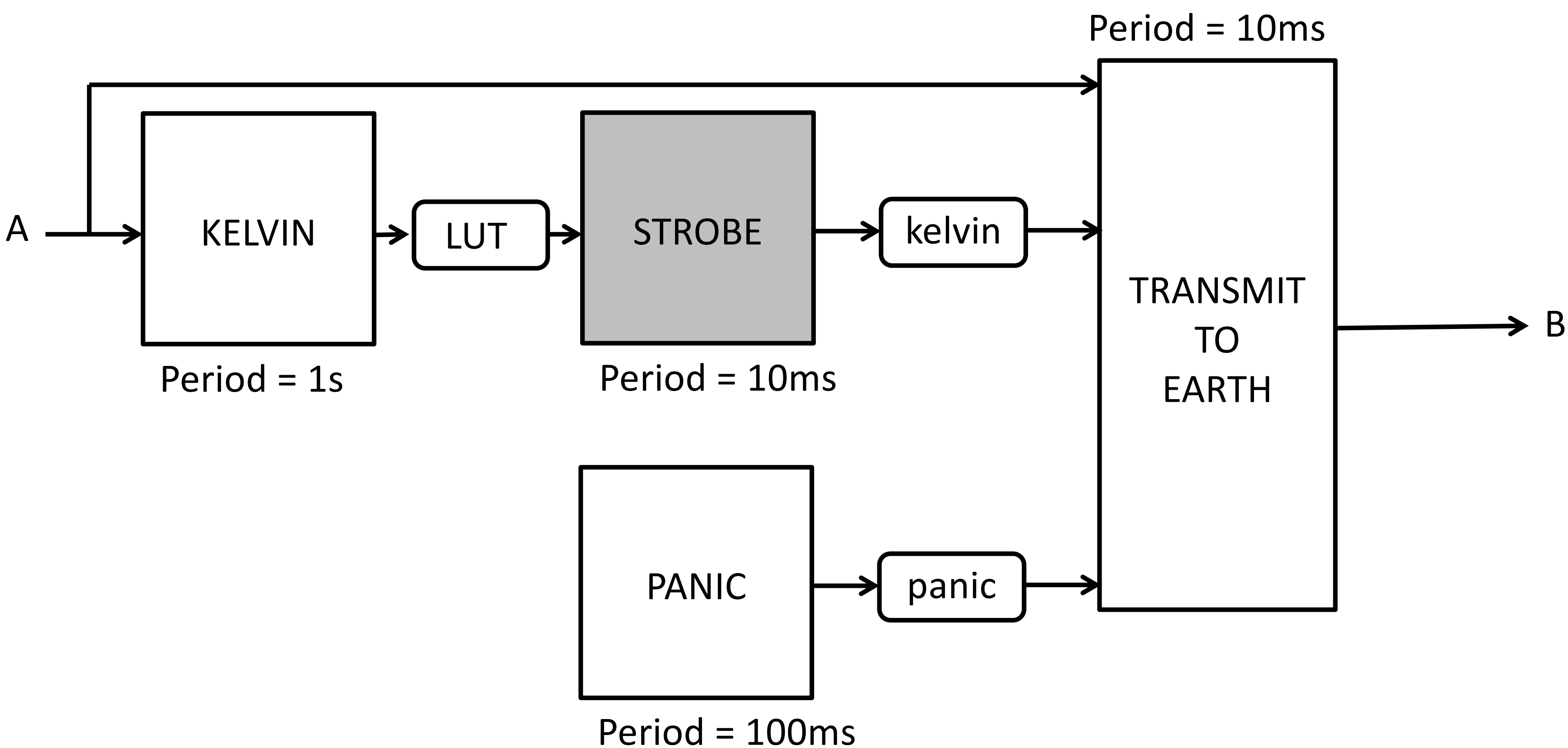


KELVIN Task

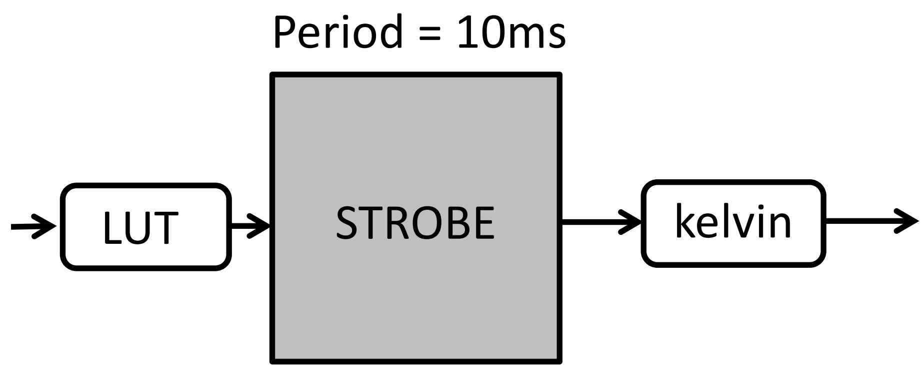


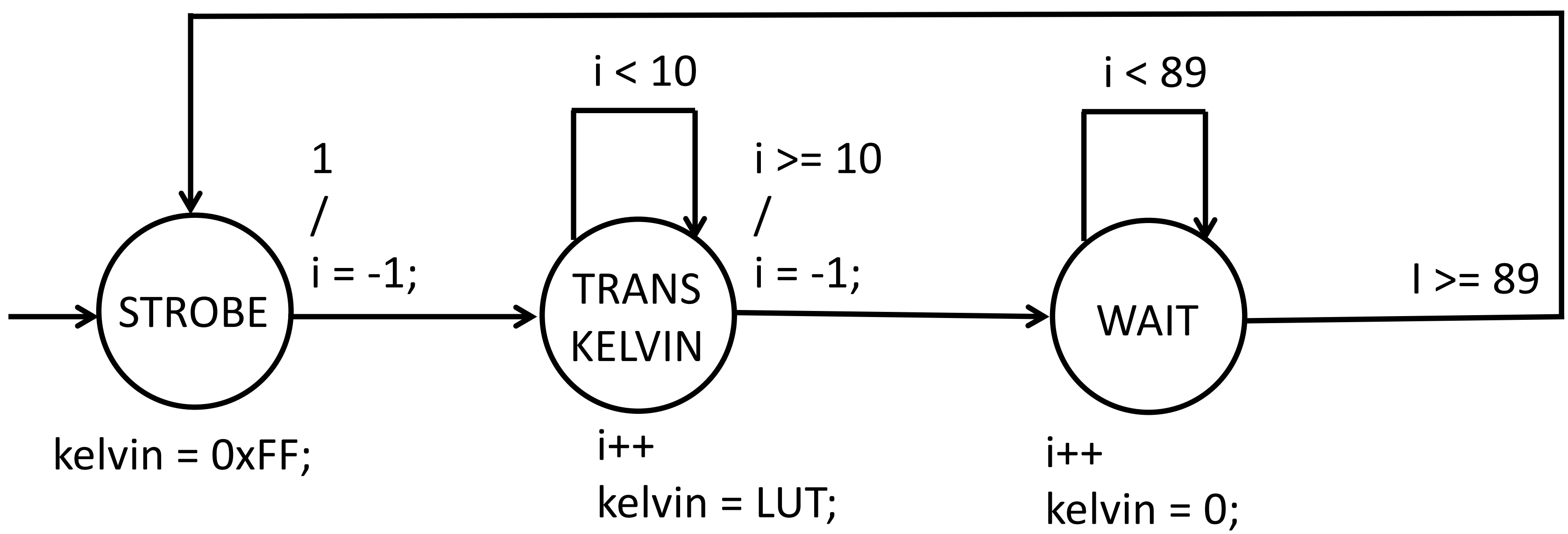


Solution #4
Concurrent SynchSM w/Shared Variables



STROBE Task



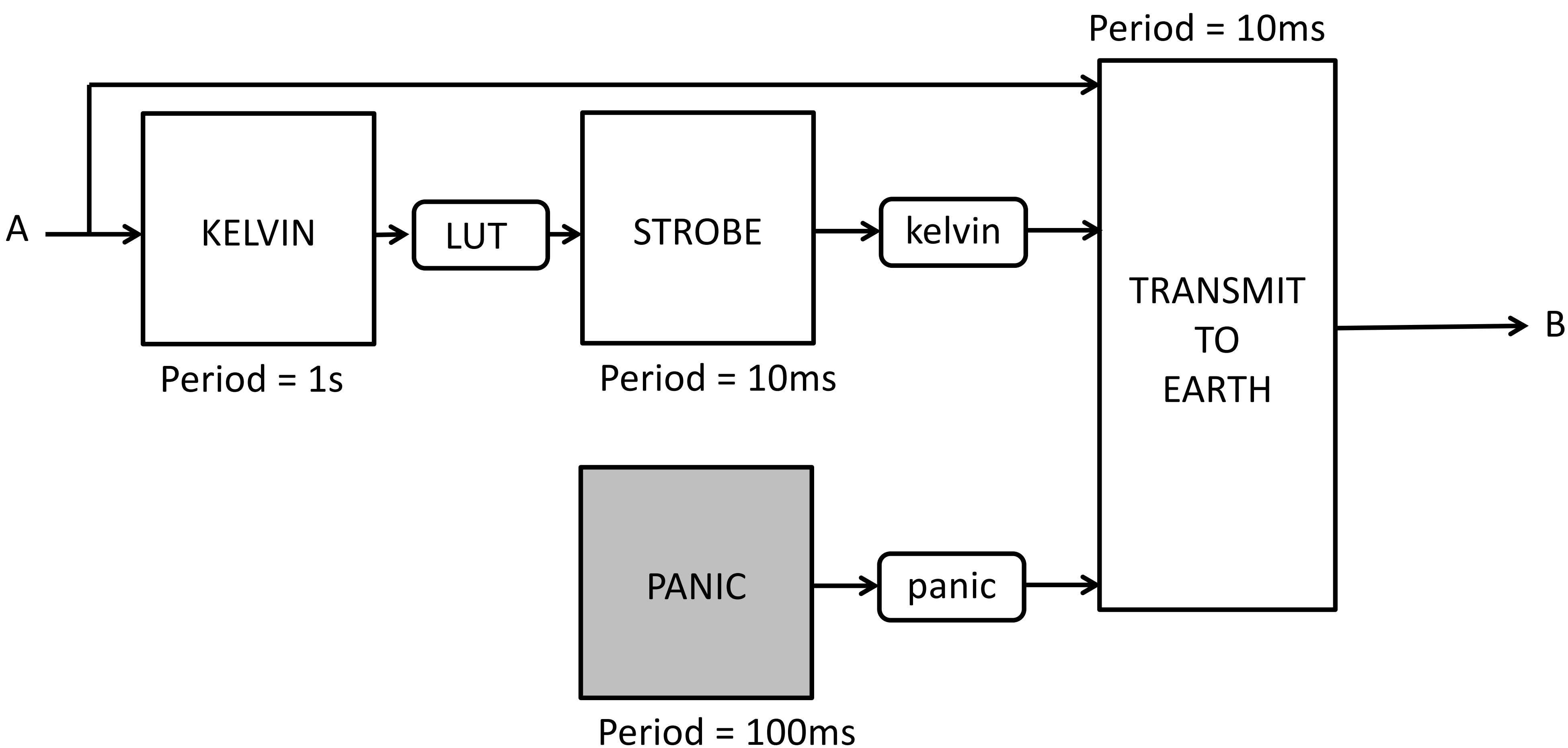


// Strobe for 10ms

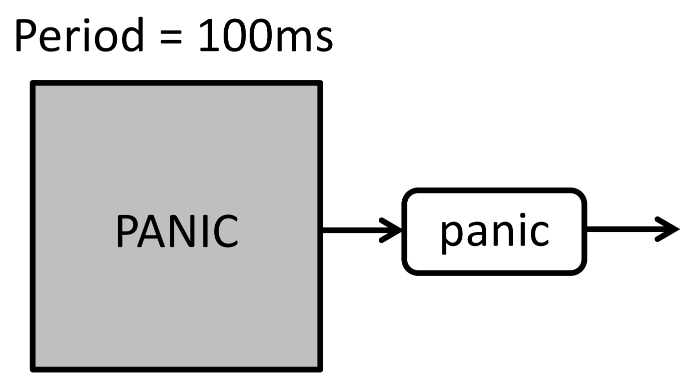
// Transmit for 100ms

// Wait for 890ms

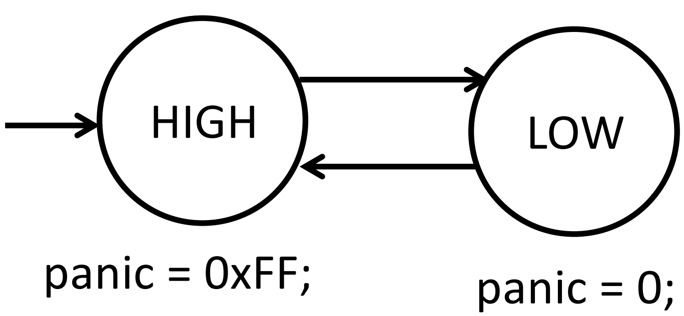
Solution #4
Concurrent SynchSM w/Shared Variables



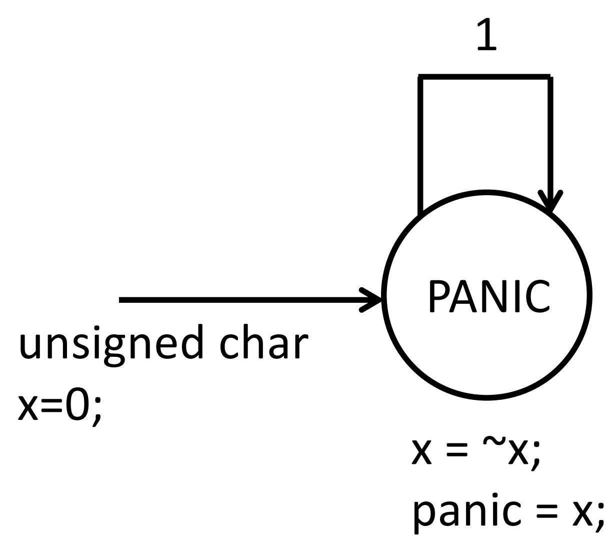
PANIC Task



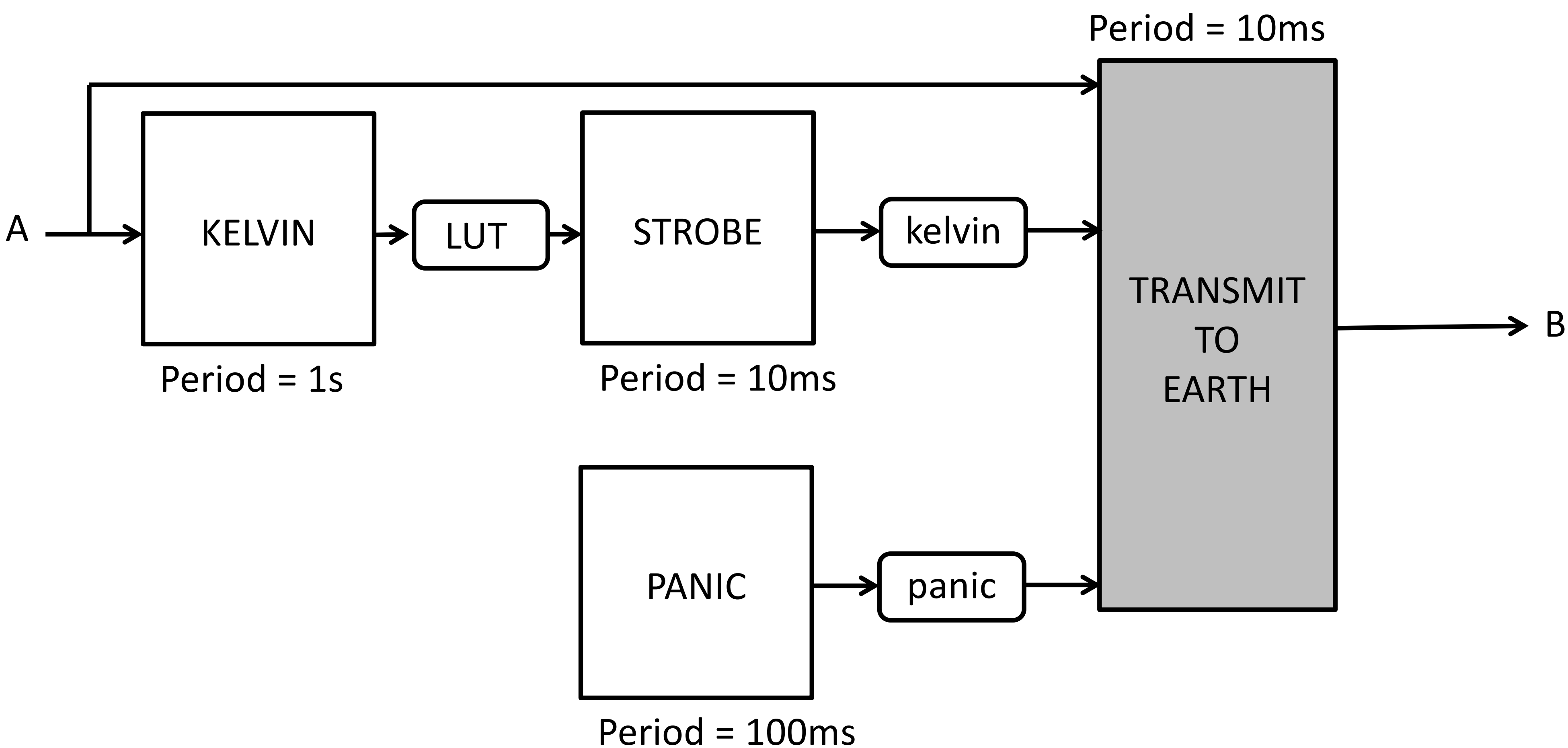
2-state Implementation



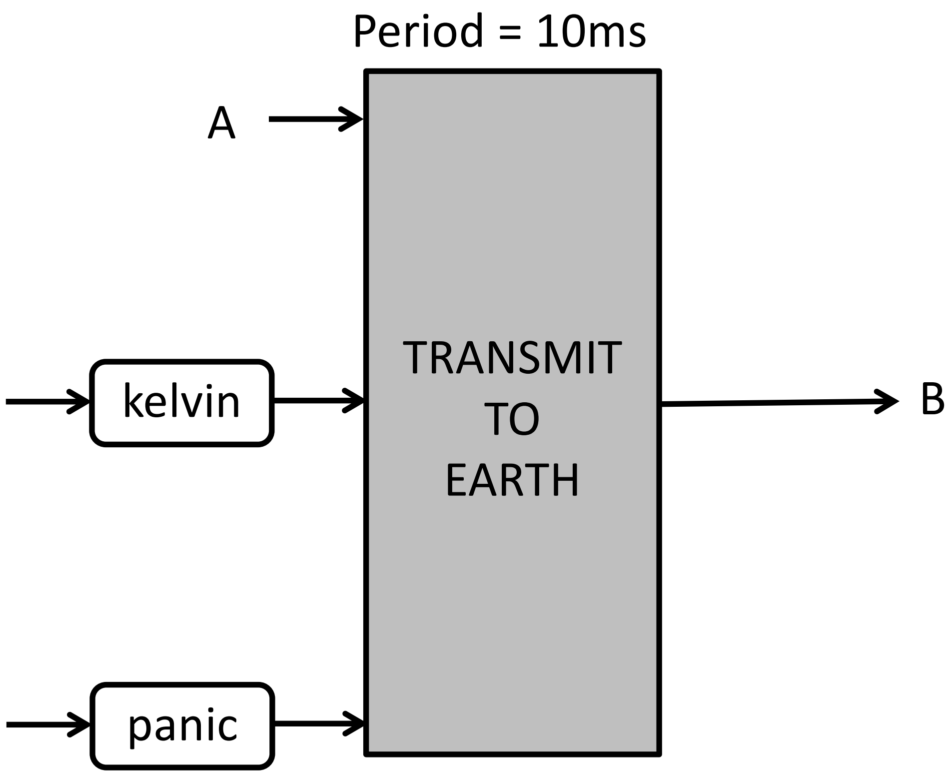
1-state Implementation

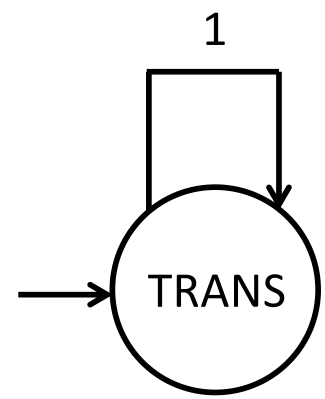


Solution #4
Concurrent SynchSM w/Shared Variables



TRANSMIT TO EARTH Task





B = (A == 0) ? panic : kelvin;

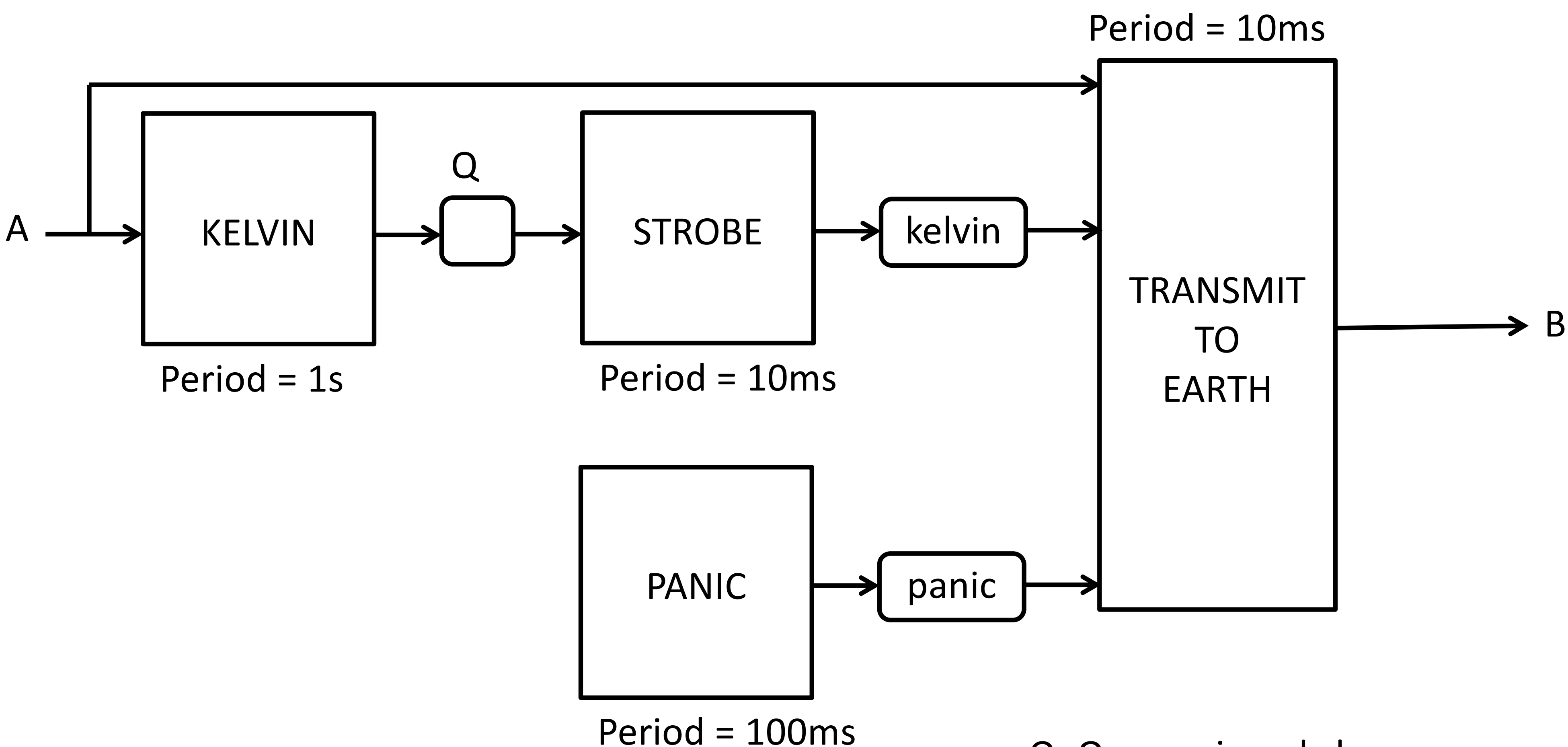
Solution #5
Concurrent SynchSM
w/Shared Variables and a Queue

- Sample input A every 1 s and put the result into a queue, rather than a shared variable

- The STROBE task no longer needs to track waiting time (890ms in Solutions #1 - #4)

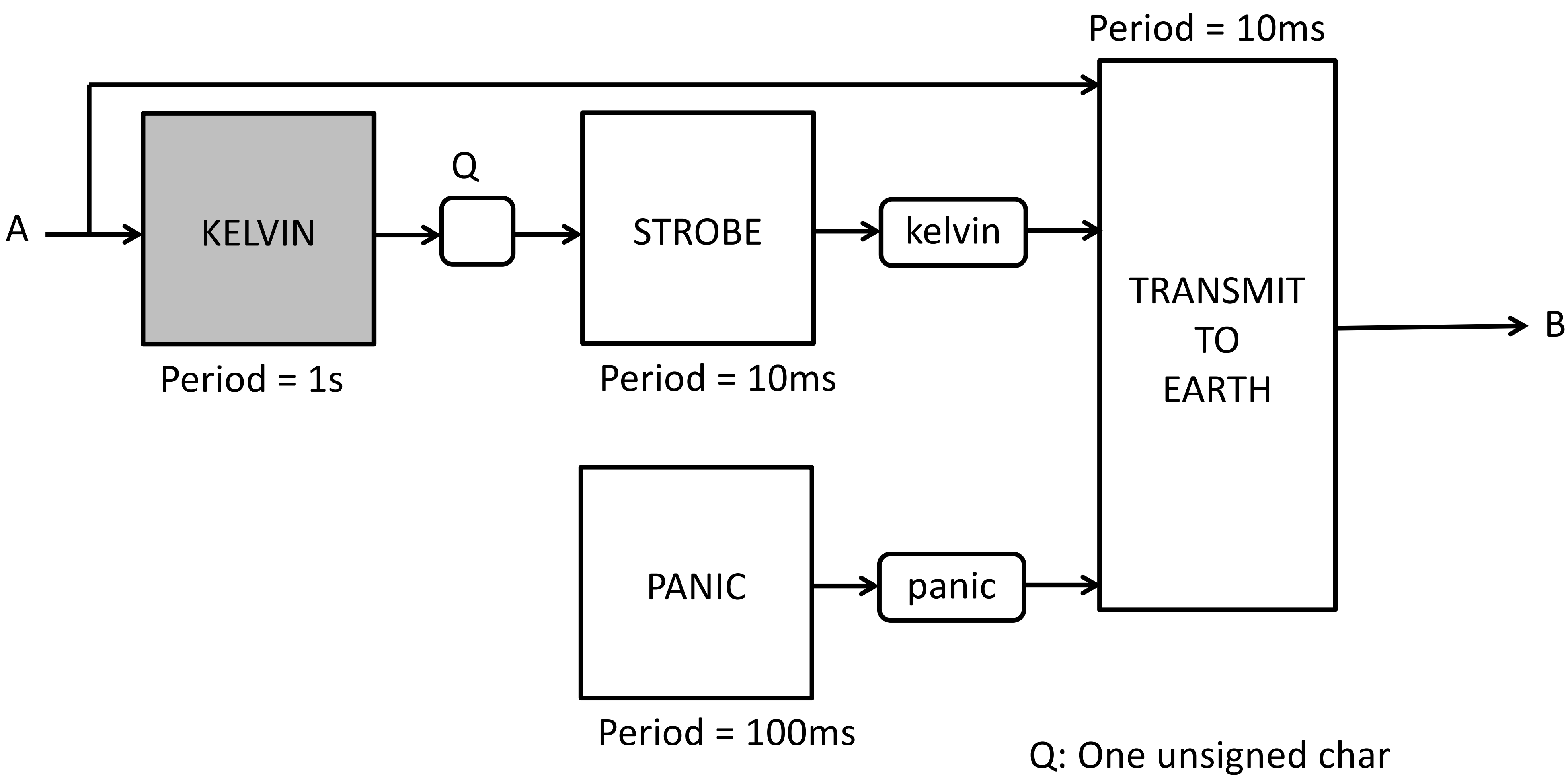
- The STROBE can simply poll the queue and wait until data is available to process. This simplifies the control logic significantly.

Solution #5
Concurrent SynchSM w/Queue

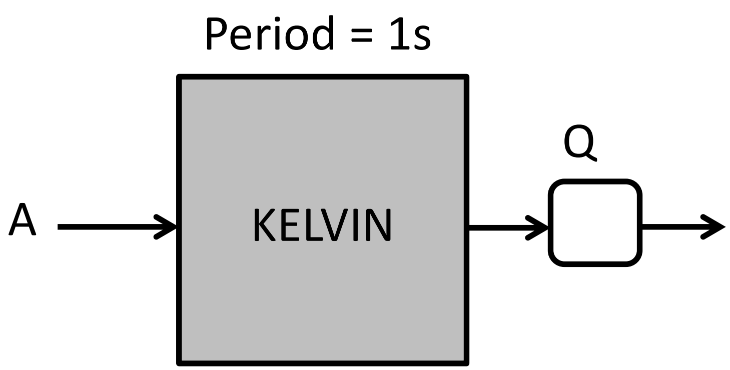


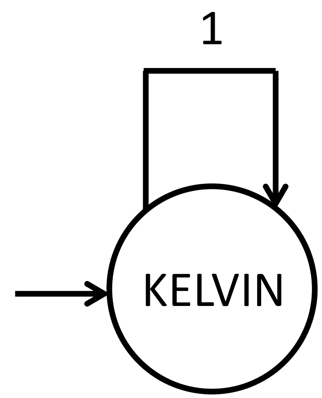
Q: One unsigned char

Solution #5
Concurrent SynchSM w/Queue

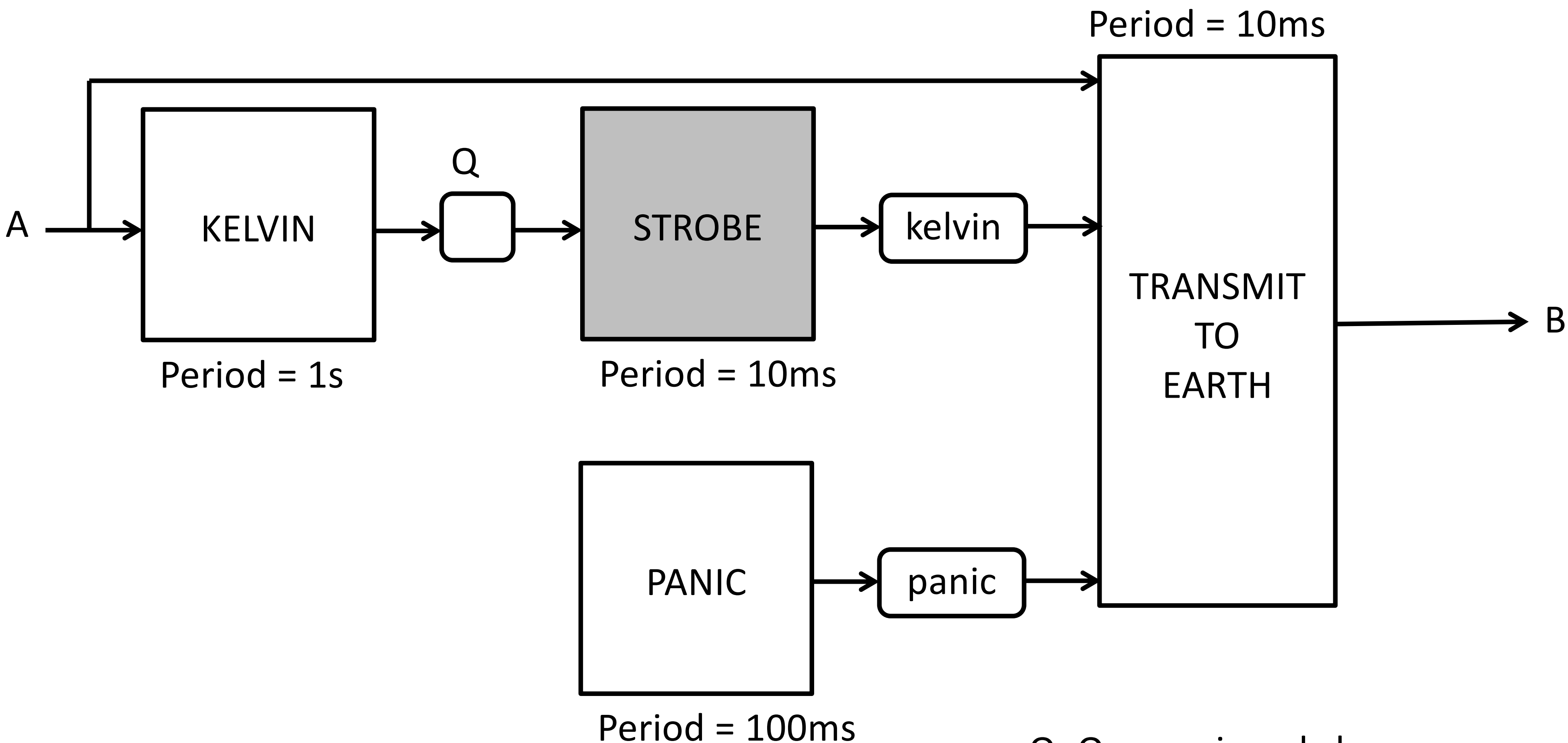


KELVIN Task



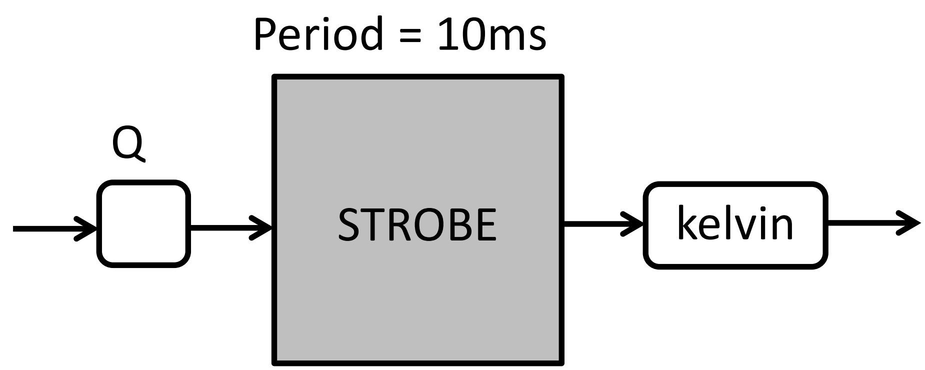


Solution #5
Concurrent SynchSM w/Queue

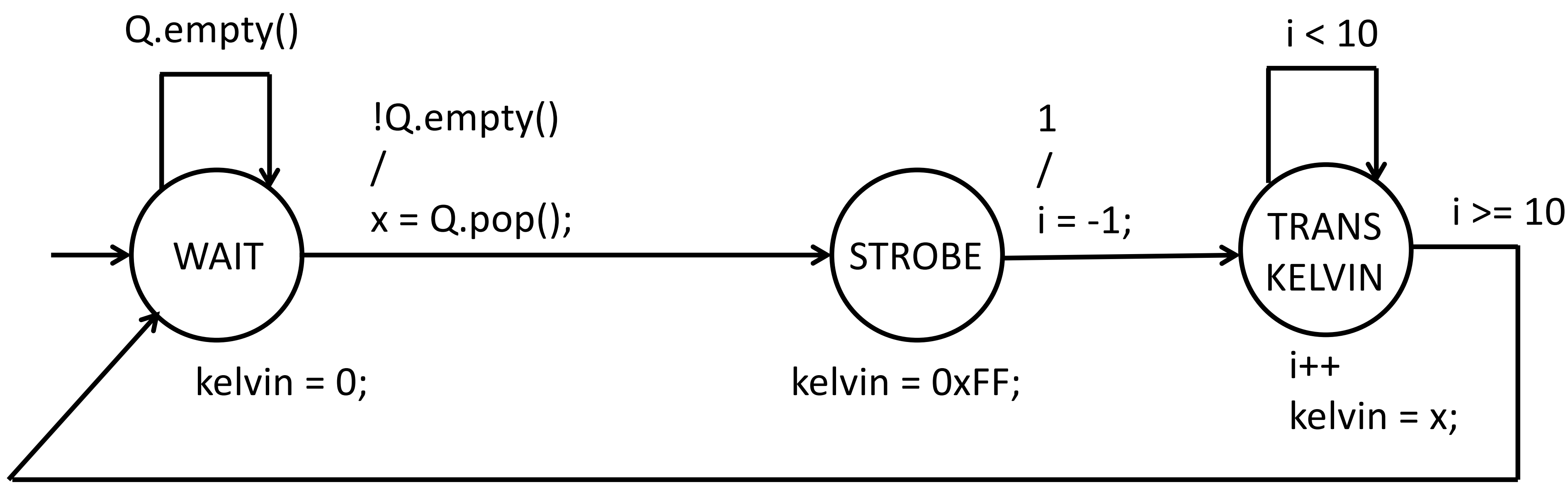


Q: One unsigned char

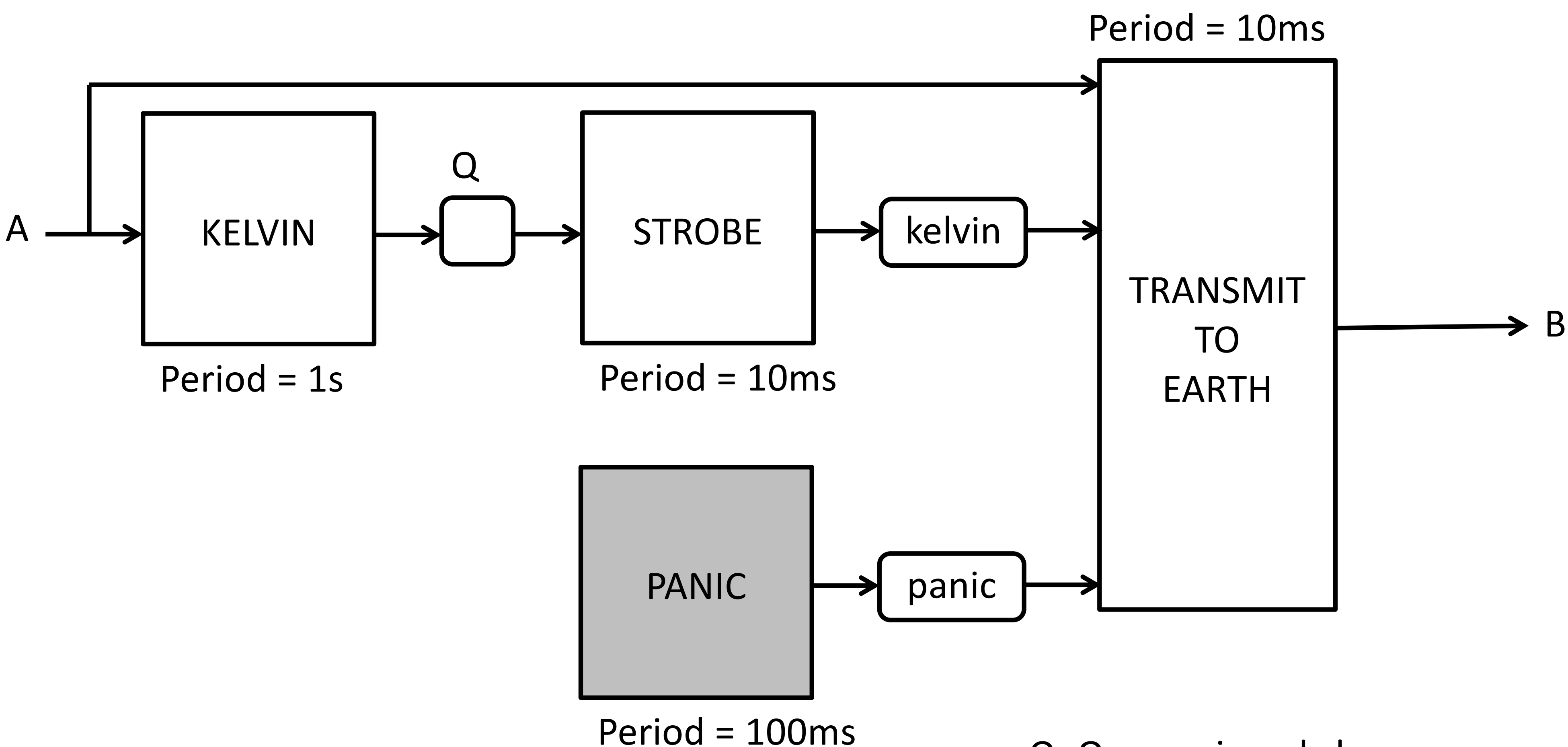
STROBE Task



unsigned char x;

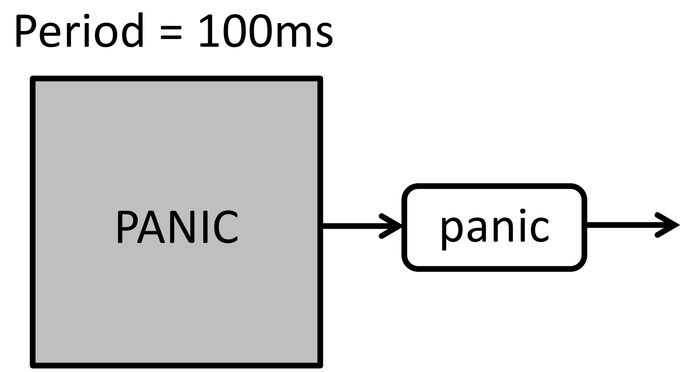


Solution #5
Concurrent SynchSM w/Queue

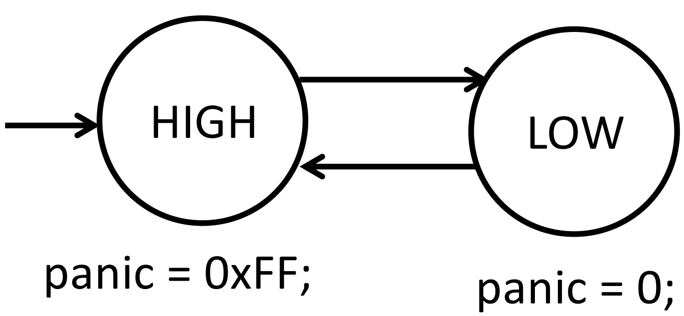


Q: One unsigned char

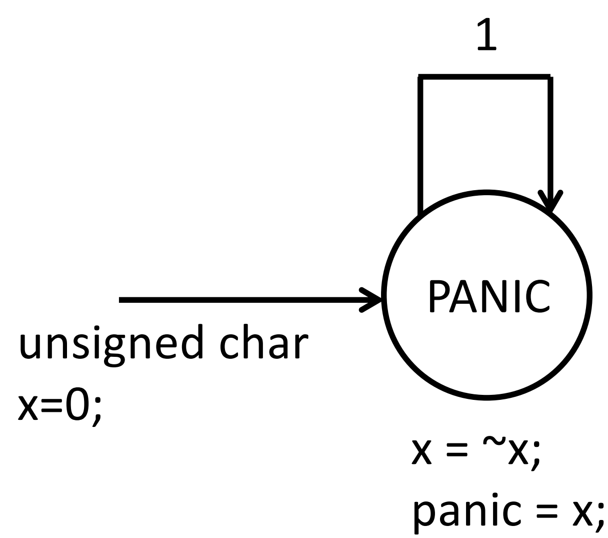
PANIC Task



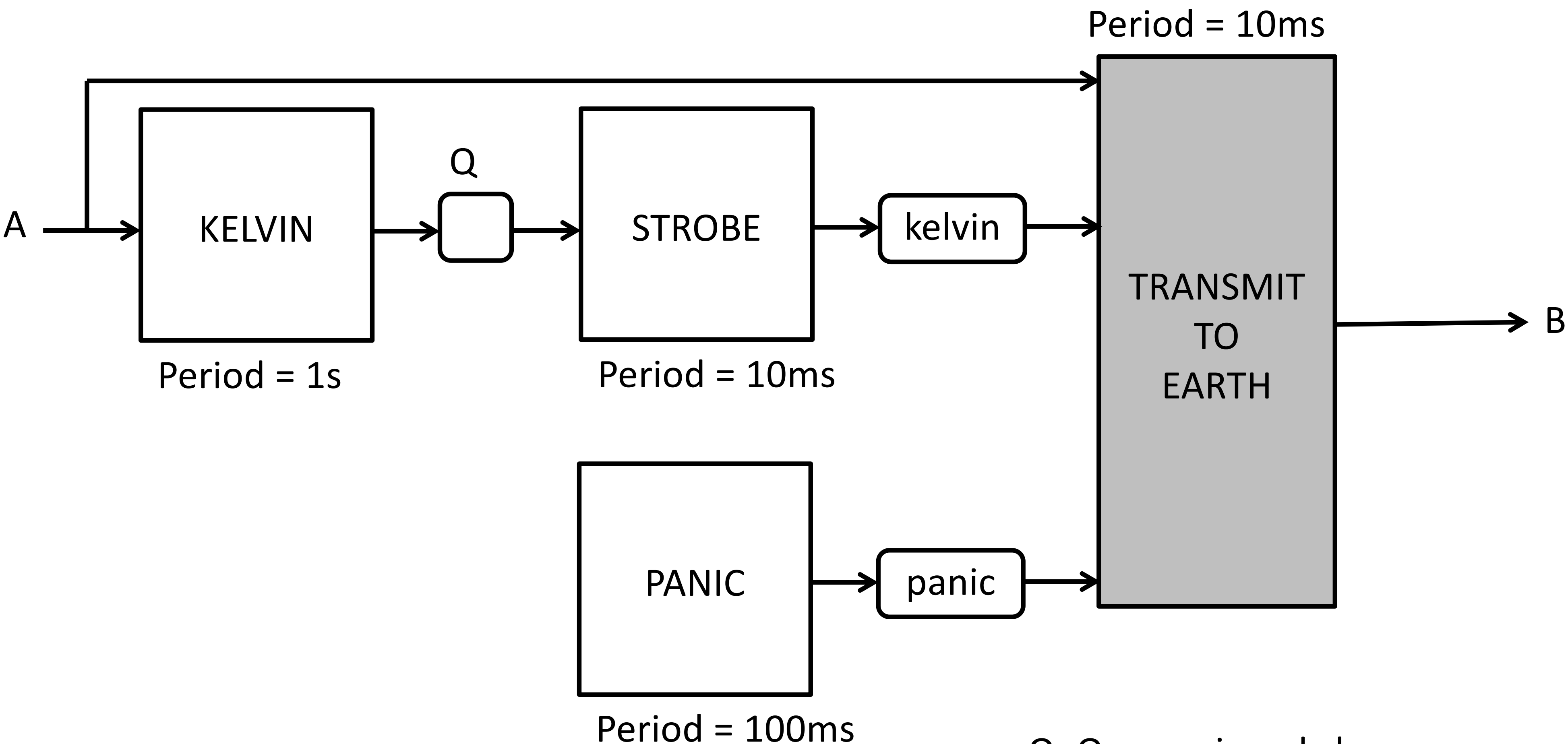
2-state Implementation



1-state Implementation

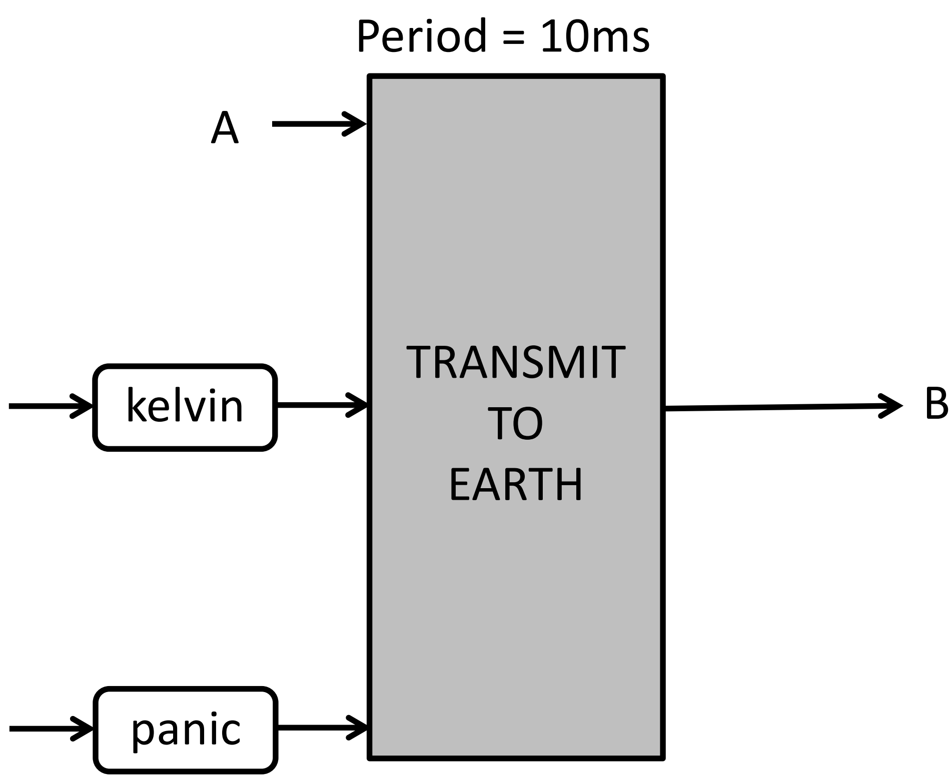


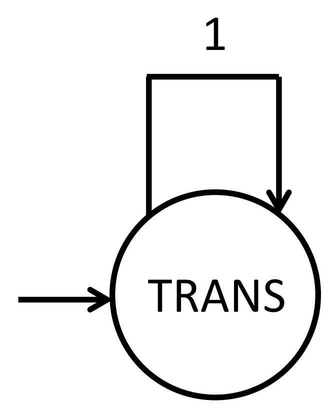
Solution #5
Concurrent SynchSM w/Queue



Q: One unsigned char

TRANSMIT TO EARTH Task





B = (A == 0) ? panic : kelvin;

Conclusion

To get the simplest overall design:

- Concurrent task decomposition
  - Separate “Transmission” and “PANIC” functionalities
  - “STROBE” and “PANIC” tasks compute their output every tick; “TRANSMIT” selects which one to output to B
  - This is far simpler than switching between “STROBE” and “PANIC” modes of operations in a non-concurrent synchSM

- Uses both a queue and shared variables
  - Only one task writes to the queue in this case
    (unlike the Floating Beacon example)

And the Moral of the Story Is…

- Queues make ~~everything~~ many things better
- Avoid “flashing” the value of a shared variable for a small number of ticks, as this requires complex cross-task synchronization