# Health/State Indicator
# OR
# Sequence Generator
# (For Embedded Systems)

## *Introduction*

This Sequence Generator or Health/State Indicator header file (s_seq.h) is developed for Arduino devices. Health/State Indicator is a feature you can implement in your embedded system. If you have a serial receiver, you can print current state/health bit through serial port and read it. Using this, we can assess the working of embedded system. But if we don't have serial receiver or display module, it is hard to know what is happening inside the system. Implementing individual LED for each state is wastage of I/O pins. So, it would be better to identify current status through single LED or buzzer (via series of 1's and 0's. i.e., unique sequences). If we are using time division duplexing for displaying each state's health, we can effectively convey the message/status of embedded system. The Sequence generator as the name states, can output data stored in an array through a digital pin. Generic functions are not much popular for this purpose. It might be because of easiness to write a new function. However, I am trying to explain both cases below. This won't be recommended as health monitoring if status varies too frequently.

## *Requirement identification*

Since we are focusing in embedded systems, it might have limited processing power. So, processing power should be utilized so efficiently. Usage of delay functions are wastage of processing power. Consider blinking LED (or LEDs) in a particular manner. Some examples with and without this header file.

| Blinking an LED in the sequence of 1011010 | |
|---|---|
| Conventional method (there might be better options available) | Using this header file |
| *void setup()*<br>*{*<br>*pinMode(led_pin,OUTPUT);*<br>*}*<br>*void loop()*<br>*{*<br>*digitalWrite(led_pin, HIGH);*<br>*delay(state_time);*<br>*digitalWrite(led_pin, LOW);*<br>*delay(state_time);*<br>*digitalWrite(led_pin, HIGH);*<br>*delay(state_time*2);*<br>*digitalWrite(led_pin, LOW);*<br>*delay(state_time);*<br>*digitalWrite(led_pin, HIGH);*<br>*delay(state_time);*<br>*digitalWrite(led_pin, LOW);* | *s_seq led_blink(led_pin, 7, 1, state_time); //this constructor is for sequence alone*<br>*void setup()*<br>*{*<br>*led_blink.init();*<br>*led_blink.reset_data();*<br>*led_blink.put_data(1,0b1011010);*<br>*}*<br>*void loop()*<br>*{*<br>*while(1)*<br>*{*<br>*led_blink.play();*<br>*//your codes here*<br>*}*<br>*}* |

| | |
|---|---|
| *delay(state_time);*<br>*//your codes here*<br>*}* | |
| | |

| Beeping a buzzer for 2 times (if some specific conditions are is met) | |
|---|---|
| Conventional method (there might be better options available) | Using this header file |
| *void setup()*<br>*{*<br>*pinMode(led_pin,OUTPUT);*<br>*}*<br>*void loop()*<br>*{*<br>*if(specific condition)*<br>*{*<br>*while(no_of_times_to_beep)*<br>*{*<br>*digitalWrite(buzzer_pin, HIGH);*<br>*delay(state_time*2);//if ON time is more*<br>*digitalWrite(buzzer_pin, LOW);*<br>*delay(state_time);*<br>*digitalWrite(buzzer_pin, HIGH);*<br>*delay(state_time*2);//if ON time is more*<br>*digitalWrite(buzzer_pin, LOW);*<br>*delay(state_time);*<br>*no_of_times_to_beep--;*<br>*}*<br>*}*<br>*//your codes here (will execute only after completion of beep*<br>*}* | *s_seq buzzer(buzzer_pin, 6, 1, state_time); //this constructor is for sequence alone*<br>*void setup()*<br>*{*<br>*buzzer.init();*<br>*buzzer.reset_data();*<br>*}*<br>*void loop()*<br>*{*<br>*if(specific condition)*<br>*{*<br>*buzzer.put_data(1,0b110110, no_of_times_to_beep); //ontime, off time can be controlled (in multiples of //state_time)*<br>*buzzer.reset();*<br>*}*<br>*buzzer.play();*<br>*//your codes here (won't be blocked by the buzzer beep)*<br>*}* |

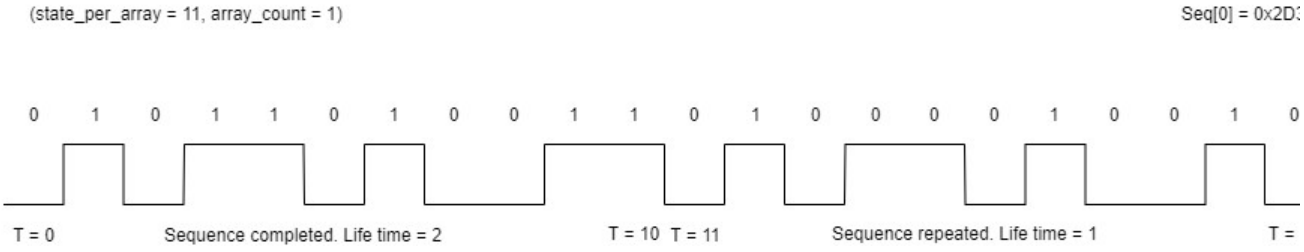
<table>
<tr><td colspan="2">Displaying status of microcontroller/embedded system</td></tr>
<tr><td>Conventional method (there might be better options available)</td><td>Using this header file<br>Considering Device 1 on/off status is available in dev_1_sts, Temperature ok/nok status in temp_sts.<br>This is a simple example where slot 1 is used for device 1 status and slot 2 is used for temperature status.</td></tr>
<tr><td>Various methods are available. I don’t know which one is better.</td><td>s_seq health(led_pin, 8, 2, 0, state_time,20);//better to use //state_time 1000 or more here<br>void setup()<br>{<br>health.init();<br>health.reset_data();<br>}<br>void loop()<br>{<br>if(dev_1_sts == HIGH)<br>health.put_data(1,0b00000100); //one blink – device on<br>else<br>health.put_data(1,0b00000101); //two blink - device off<br>if(temp_sts == HIGH)<br>health.put_data(2,0b00110000); //short blink – temp ok<br>else<br>health.put_data(2,0b00111110); //long blink – temp nok<br>health.play();<br>//your codes here<br>}</td></tr>
</table>

Health monitoring of an embedded system implementation is shown in last example. This won’t be recommended if status varies too frequently. Example starts with object creation. The constructor defines parameters for the sequence. Here, eight states are defined for one slot/array. Two slots/arrays are defined (Two status need to be displayed). So, each array element can be considered as one slot and can be identified for monitoring a unique health/state parameter. Dummy state is defined as 20 i.e., after displaying these two slots (8 states * 2 array elements = 16 states), 25 extra dummy states will be added before restarting the sequence.

In the example first slot/first array element slot is allotted for device 1 health. If one blink appears in this slot, it means device 1 is working. If two blink appears, it means device 1 is not working. Second slot is allotted for temperature status. Short blink in this slot means temperature ok. Long blink in this slot means temperature not ok (might be upper or lower threshold). If some complicated identification patterns are used, both statuses can be displayed in same slot. But it is not recommended.

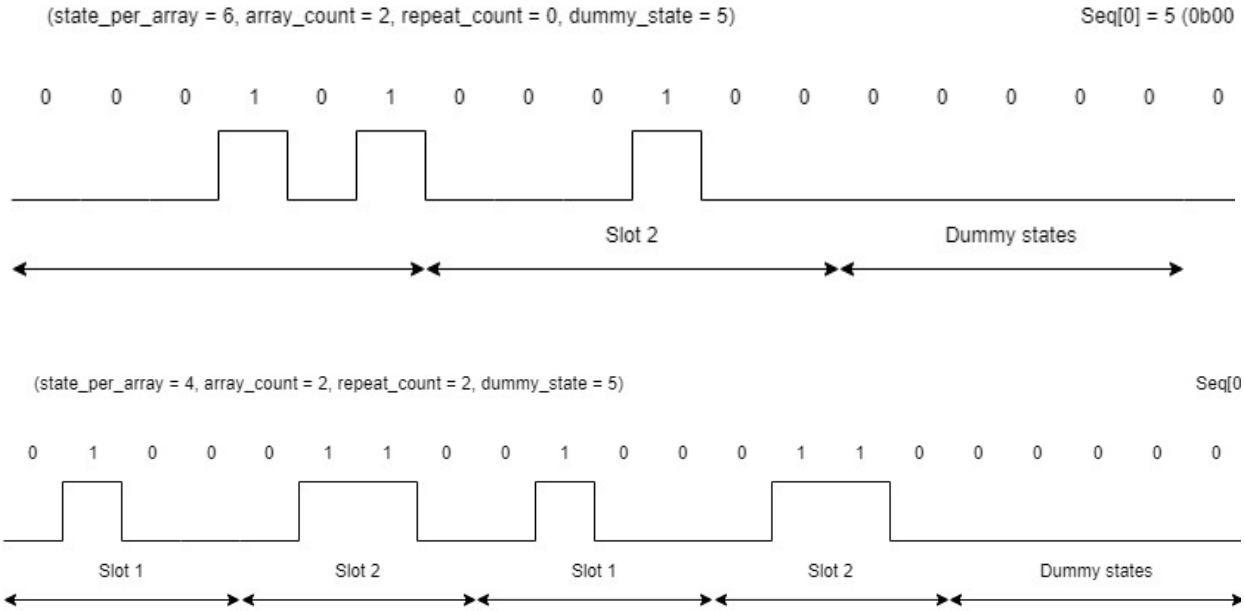
Object.play() function is used to control the current status of LED. It will automatically assess the value in that instant as per the sequence.

## *Working diagram*




One sequence is shown in the figure. Here state_duration is taken as time required for completing one state. If it is a complicated sequence, we need to take a base time (multiples of this state_duration must be capable of representing any state in the sequence). Total state in the sequence is 11. So, we can use state_per_array as 11. If total state is greater than 32, we need to split the sequence and make it pieces of sequences less than 32. Consider a sequence of 70 states. Then it can be split as 7 array elements (seq[0] - seq[6]) of 10 state each. T is a representation of time slot.

All the sequences are never ending. In order to make a sequence repeat for 'n' times, define the lifetime with 'n'. This can be done while entering the data (put_data();). Don't confuse with the term repeat_count and lifetime. In case of a normal sequence (not health monitoring/status monitoring), repeat_count and dummy_state_count are not required. If repetition is required, define that count as lifetime. Figure shown above represents a sequence with lifetime of 2. If lifetime is not defined/lifetime is zero, it won't stop at T=31.




Now consider the health monitoring/status monitoring (figures shown above). In case of health monitoring/status monitoring, status monitoring is required in fixed interval. If it is looping, it will be hard to identify slots (in the case of sequence generation, looping is required). In order to stop looping, it is possible to add dummy states after displaying all states. State_duration can be taken as 500ms/1000 ms for easy identification. Slots should be identified for each monitoring/status. Slot means array element. If 'n' slots are required, array count should be initialized as 'n'. Sometimes, it may not be possible to identify the slot. So, it is a good practice to keep first slot as alert slot. By hearing/watching that sequence (alert sequence), user can

wait for the actual health status. If dummy states are too large (total duration taken by dummy state is greater than 10-15 minutes), this alert sequence may not be sufficient to get user attention. Since it is not advisable to miss health/status monitoring, programmer can use repeat count in order to repeat the monitoring (all slots) 'n' number of times as defined in repeat_count. In this case the repetition can be easily identifiable by adding some dummy slots.

Please note this library works as MSB first i.e., MSB of the array will be displayed/shown in the first state of respective slot/array. You can check the timing diagrams for more clarity. Note the use of repeated and restarted in the timing diagrams (in case of sequences repeat/restart doesn't have any difference as count_repeat is not involved).

## *Function calls*

All available function calls are mentioned below

1. s_seq(int t_pin, int t_state_per_array, int t_array_count, int t_count_repeat, unsigned long int t_state_duration, unsigned long int t_dummy_state_count);
    - This is constructor. This is useful for health/status monitoring. For sequence alone, use the other constructor.
    - Pin – Output pin to generate sequence
    - State_per_array – No. of states defined in one element of seq[] (for one slot/array).
    - Array_count – No. of slots allotted. Including dummy slots for repeating.
    - Count_repeat – No. of times it needs to be repeated, if required. '0' if no repetition is required.
    - State_duration – Duration of a single state.
    - Dummy_state_count – If dummy states are required before sequence restarting (not to be confused with dummy slots which helps differentiate between repeating). Otherwise use '0'.
    - There is a difference between dummy slots (used before repeating) and dummy states (used before restarting). If count_repeat is '0', both will work as same.
2. s_seq(int t_pin, int t_state_per_array, int t_array_count, unsigned long int t_state_duration);
    - This is constructor. This is supposed for sequences alone.
    - Pin – Output pin to generate sequence
    - State_per_array – No. of states defined in one element of seq[] (for one slot/array)
    - Array_count – No. of slots allotted. Including dummy slots restarting/repeating. Since count_repeat is not applicable here, dummy slots (if required) must be put here.
    - State_duration – Duration of a single state.
3. void init();
    - This should be called after object creation. It will initialize all seq[] and state pin as output.

4. void play();
    - This is the function which assess the time and change the state of ‘pin’. You need to put this function in the main loop. If some loop is blocking this function call or if the program needs more time to reach this point, the sequence/health monitoring won’t work properly. In that case increase the state_duration.
5. void reset();
    - Reset the sequence. You can call this if you are going to start a data slot with lifetime.
6. void put_data(byte t_seq_no, unsigned long int t_data);
    - This function can be used for defining values to a slot (seq slot).
    - The lifetime will be infinite i.e., the defined value won’t change unless it is cleared or change by user.
    - This method can be used for health/status monitoring and endless sequence.
7. void put_data(byte t_seq_no, unsigned long int t_data, byte t_lifetime);
    - This function can be used for defining values to a slot (seq_slot).
    - The lifetime means, the validity of this seq_slot. After each sequence restart, lifetime will be decreased and data will be cleared.
    - This method can be used for beeping a buzzer or blinking an LED for ‘n’ times.
8. void clear_data(byte t_seq_no);
    - This function can be used for clear data of a slot (seq_slot).
9. void show_data();
    - This will serial out the data of slots/arrays (seq_slots).
10. void reset_data();
    - This will reset whole data in slots/arrays (seq_slots).

## *Conclusion*

This guide is for understanding usage of s_seq.h. If you have suggestions or for reporting bugs, contact muralysunam@gmail.com. Example programs attached.