Pawel Gburzynski

AS3932 Low Frequency Wakeup Receiver

driver

May 3, 2015

Introduction

AS3932 is a low frequency (125 kHz) receiver intended for lop-power remote wakeups or RFID emulation. Its maximum current consumption is less than 6µA (continuous listening on all three channels) with various savings possible down to below 2µA.

Present functionality of the driver

For now, I have only implemented the functionality required for AP331 of Alphatronics. While it would be relatively easy to add options, I didn't want to unnecessarily inflate the code for things that may never be needed. This will come later, if a need arises.

The (external) transmitter sends a single-pattern signal compatible with the default settings of the module. The pattern is also default (0x96), which means that it doesn't gave to be changed. Any deviations from those standard settings will require a modification of the driver, which may come as static (compiled in) or dynamic options.

The data following the triggering pattern consists of 5 bytes: 4 data bytes + 1 parity byte. The first three bytes encode the device Id (MSB comes first), the last byte is a special code settable with two rotary switches on the transmitter.

The driver reacts to the correct preamble + pattern + data sequence with a valid parity code and returns to the application the 4 proper data bytes as a sensor value. Owing to the fact that when the receiver is present in the proximity of a transmitter, there will be a continuous supply of events, it is recommended to disable the device for a while after receiving one event. Here is the recommended way to interact with the sensor:

```
fsm as3932_sensor_monitor {

        state WAIT_EVENT:

                as3932_on ();
                wait_sensor (SENSOR_AS3932, SENSOR_EVENT);

        state SENSOR_EVENT:

                read_sensor (WNONE, SENSOR_AS3932, &val);
                as3932_off ();
                delay (SOME_DELAY, WAIT_EVENT);
}
```

The functions as3932_on and as3932_off are defined by the driver. If the sensor is on, and an event is received before wait_sensor has been issued, the event remains pending, so wait_sensor will immediately unblock and subsequent read_sensor will return the data. Note that val should be a 4-byte structure, e.g., lword. If all four returned bytes are zeros, it means that there is no event to report (read_sensor never blocks).

Here is a sample data value returned into a lword (in hex):

```
5439B200
```

Assuming a little-endian architecture, and considering that the bytes are written MSB-first, they should be read as: device Id = 0x00B239 (45625), rotary dials = 0x54.

Note that if the device remains on while events are not being monitored (say, we remove the on/off operations from the above loop), then the first event that shows up within that

interval will block the reception of subsequent events until it is eventually read (by read_sensor). Thus, the reported event may not be recent. It is enough to turn the device on (without turning it off first) to erase the last (possibly old) event (so the call to as3932_off could be removed from the loop without affecting its formal functionality), but turning the device off makes better sense, because it then consumes (practically) no power.