Фишка в следующем:

Код датчика в оригинальном алгоритме первоначального распознавания кодов всех подключенных датчиков, по блоксхеме из AppNotes читается сначала во временный буфер, а не сразу в массив кодов. Только после определения кода каждого следующего датчика он копируется из буфера в следующую строку массива. Буфер при этом не обнуляется перед считыванием следующего кода.

Делается это для реализации блока алгоритма для условия, когда положение текущего конфликта ответов датчиков (индекс точки принятия решения) не дошло до положения последней точки принятия решения при считывании кода предыдущего датчика. В этот момент нужно проверить, что было записано в текущем бите кода у предыдущего датчика.

- Если там был 0, то данный бит будет оставлен как есть, а текущая точка **БУДЕТ** зафиксирована, как новая.
- Если там была 1, то данный бит также будет оставлен как есть, но текущая точка **НЕ БУДЕТ** зафиксирована, как новая точка принятия решения (новый конфликт).

Во многих реализациях этого алгоритма в кодах из сети этот момент упущен, пишут код прямо в массив. В результате эта ветка алгоритма по факту не работает, поскольку для каждого нового датчика в новой строке массива текущий бит всегда будет 0. В него ничего не было ещё записано, а сам массив перед стартом поиска обнулён. Из-за этого программа не может распознать больше двух датчиков.

Как выход из ситуации при записи кода сразу в новую строку массива, при попадании в эту ветку алгоритма нужно проверять не текущий бит текущей строки массива, а бит с этим индексом, но из предыдущей строки. Именно этот вариант реализован в данном коде.

Фрагмент алгоритма, о котором идёт речь, обведён красным на блоксхеме.

ROM BIT (ROM BIT INDEX), обведённый синим, это и есть текущее значение бита в буфере, оставшееся от считывания кода предыдущего датчика.

- ROM BIT – значение текущего бита в буфере
- ROM BIT INDEX – текущий индекс бита.

ROM SEARCH Figure 5–3