Design of the Ripple UID Algorithm

– Abhijit A. Bokil

Contents

Overview

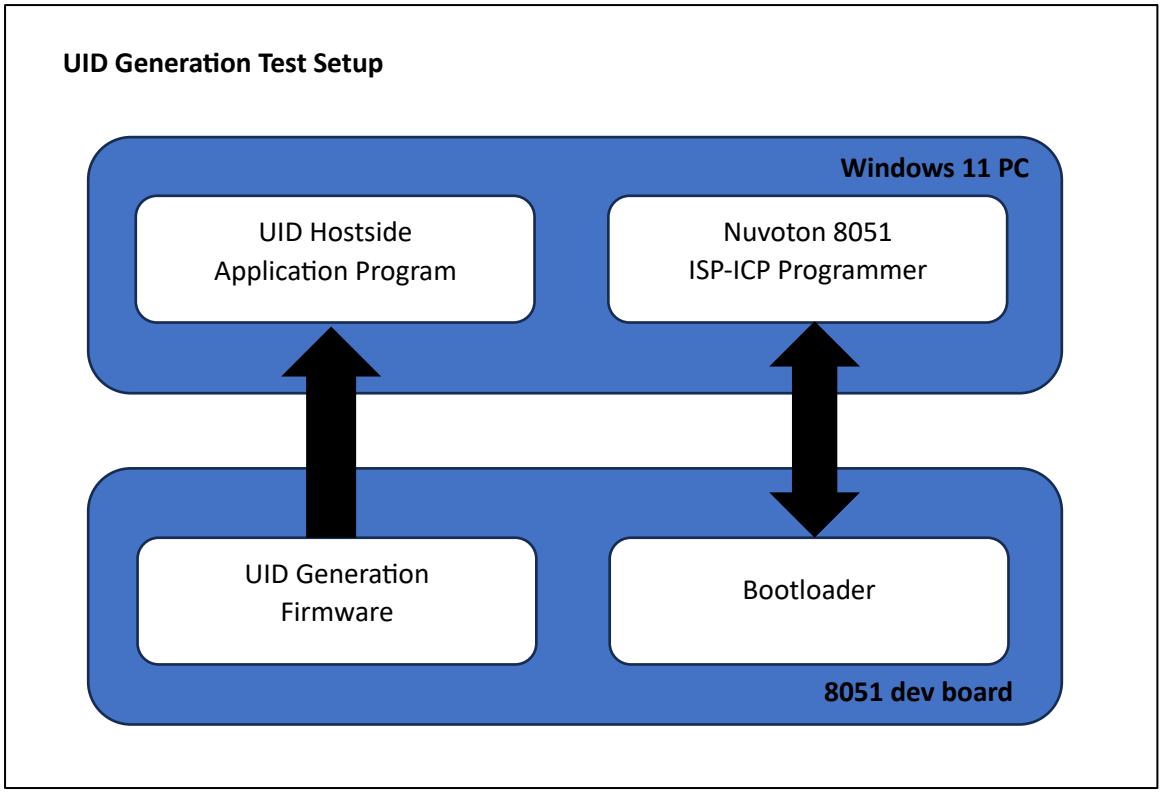
It is required to build firmware that automatically generates unique IDs based on specified criteria while ensuring randomness, uniqueness and scalability.

(Please refer to the associated Research Report for an overview of unique ID generation.)

The following specific case was chosen for the project.

- Software Language: C(C17)
- Compiler: SDCC (target), GCC (hostside)
- Development environment/system: VSCode on Windows11
- Target Hardware: Development board with a Nuvoton W78E052DDG (8051/8052 variant) microcontroller running at ~40Mhz (plus onboard DS1307 RTC chip), on-chip 8K Application Program ROM (Purchasable at https://nskelectronics.in/8051%20DEVELOPMENT%20BOARD)
- Host/test system: Windows 11 PC

Test Setup



Note that the UID is generated on the target board and the test host PC does not play any role in this process. After the UID is generated by the firmware, it writes the UID to the onboard serial port (UART) and the hostside C program merely reads and displays the UID.

In practice, the UID is permanently written to ROM on the device before it ships out.

Specified criteria

The following input parameters are supplied to the firmware:

1) Manufacturing data
    a. Manufacturing Week
    b. Manufacturing Month,
    c. Manufacturing Year
2) Test date
    a. Testing Year
    b. Testing Month
    c. Testing Date
3) Hardware characteristics

Additional parameters have been used in the algorithm as described later in this document

Considerations

Randomness

In order to ensure randomess, our UID must contain a significant number of randomly generated bits, say N. This will imply that ceterus paribus, the collision probability will be $1/2^N$ which must be a very low number.

Uniqueness

In order to ensure uniqueness, our UID must contain data from the above specified input criteria. The underlying assumption is that 2 different devices on the same assembly/test line will differ in their input criteria by atleast 1 bit, i.e. no 2 devices can have all inputs identical.

Scalability

In order to make the system scalable, i.e. allow the generation of a large number of UIDs, it may be advisable to generate IDs concurrently/parallelly on multiple lines. In such a case, two devices on two separate assembly lines may have all their input parameters identical. The only way they can be distinguished from each other is a set of bits that correspond to the index of the assembly line.

Additional assumptions

We would like the UIDs generated to be conveniently stored in a database in chronological order. Therefore, we will assign time data (point in time of UID assignment) to the UID in addition to the specified criteria, and the random section. Obviously, the time data should be in the beginning of the UID i.e the initial part of the UID in network byte order.

UID broad structure

Based on the above considerations, we have opted for a UID format which is 128 bits long like a typical UUID, and has the broad structure depicted below. *(*We have avoided the v3 and v5 formats because there is no visibility of unique criteria in the final generated UID. Since reversing a hash is nearly impossible, there is no 1-1 correspondence between the supplied criteria and the UID.) Our format is a hybrid of the v1 and v4 UUID formats (refer Research Report) and *does not conform to RFC4122.*

VARIABLE	FIXED	RANDOM
Time data	Specified criteria	Randomness

We examine each component of the UID in turn:

- *Time data*

  v1 UUIDs use timestamping i.e. the 60-bit time data field contains the number of 100 nanosecond intervals elapsed since midnight 15 October 1582 UTC. However, this field size was considered excessive and it was decided to adopt the timedata structure shown below:

Main Field	Time data					
Sub field	**Year**	**Month**	**Date**	**Hours**	**Minutes**	**Seconds**
Number of bits	7	4	5	5	6	6

Note that this means there is atleast a 1+ second interval between the UID generation of successive devices. (This seems a reasonable assumption to make in the real world. This is enough to allow for UID generation of 3600 devices an hour per test line.)

Thus, 33 bits have been employed for time data out of the 128 total UID bits.

For the purposes of the project the timedata is fetched by the microcontroller from the DS1307 RTC chip on the development board used via an I2C interface.

- *Specified criteria*
  All the criteria mentioned as input parameters form part of the 'uniqueness' section of the UID. Details are given in the UID format section.
  For the purposes of the project, the specified parameters have been stored in the on-chip APROM of the 8051 microcontroller.

- *Randomness*
  The rand() pseudo-random generator function is used to generate the 'randomness' section of the UID. Note that the RNG is seeded using srand(time_data_seconds).

UID format

Depicted below is the complete UID format with all fields and the number of bits per field.

RIPPLE UID																
Field	**Y**	**M**	**D**	**h**	**m**	**S**	**TL**	**DC**	**My**	**Mm**	**Mw**	**Rb**	**Ty**	**Tm**	**Td**	**R**
Number of bits	**7**	**4**	**5**	**5**	**6**	**6**	**3**	**3**	**7**	**4**	**6**	**1**	**7**	**4**	**5**	**55**

As per network byte order, the MSB is at the top and the LS bits are towards the bottom of the table

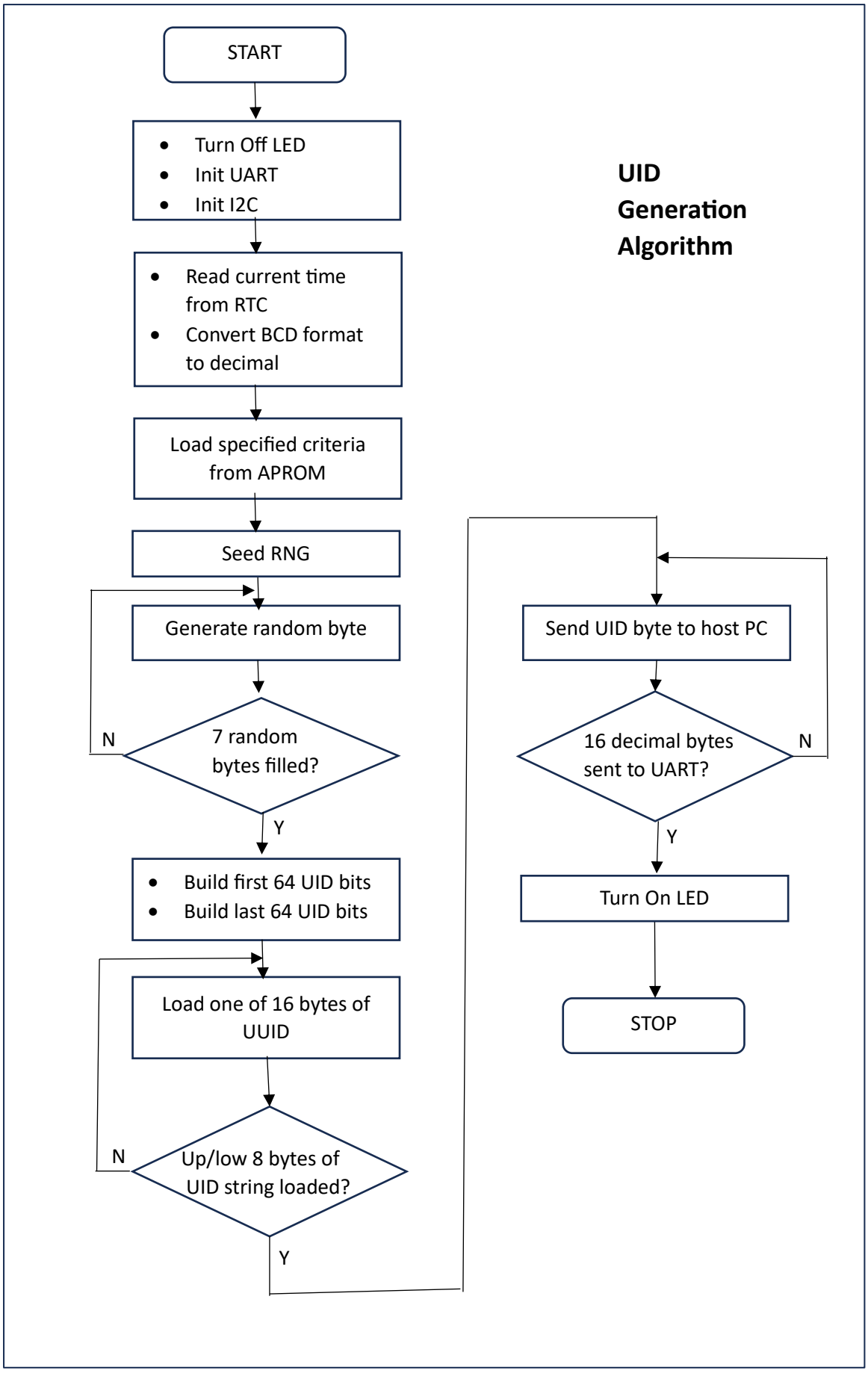
Field Key	Summary	Range
Y	Time data Year	0 – 128
M	Time data Month	0 – 12
D	Time data Date	0 – 31
h	Time data hour	0 – 24
m	Time data minutes	0 – 60
s	Time data seconds	0 – 60
TL	Test line	0 – 7
DC	Device characteristics index	0 – 7
My	Year of manufacture	0 – 128
Mm	Month of manufacture	0 – 12
Mw	Week of manufacture	0 – 53
Rb	Reserved bit	1
Ty	Year of test	0 – 128
Tm	Month of test	0 – 12
Td	Date of test	0 – 31
R	Random number	0 – (2^{55} minus 1)

Notes:

- As explained in the broad structure section, 33 bits as used here are sufficient to timestamp the device upto seconds resolution
- Provision has been made for 8 possible test lines indexed by the 3 bits
- Provision has been made for 8 possible device characteristic groups indexed by the 3 bits (e.g. 8 different PCB versions)
- The year, month and week of manufacture together are indicated by 17 bits
- Provision has been made for a bit reserved for future use. (Default value = 0)
- The year, month and date of testing together are indicated by 16 bits
- A 55 bit random number is tagged at the end of the UID
- In the interests of brevity & efficiency, the various fields do not obey byte boundaries.

Algorithm

Initially the communication interfaces (UART/I2C) are initialized. Then current datetime is read from the Real Time Clock. Specified criteria are then read from ROM. 7 random bytes (56 bits) are then generated. The UID is generated in 2 parts of 64 bits each.The 16 decimal value bytes are then transmitted over the serial interface to the test host, where the UID is displayed. The flowchart shown below implements the designed UID generation algorithm.


START
Turn Off LED
Init UART
Init I2C
Read current time from RTC
Convert BCD format to decimal
Load specified criteria from APROM
Seed RNG
Generate random byte
7 random bytes filled?
N
Y
Build first 64 UID bits
Build last 64 UID bits
Load one of 16 bytes of UUID
Up/low 8 bytes of UID string loaded?
N
Y
Send UID byte to host PC
16 decimal bytes sent to UART?
N
Y
Turn On LED
STOP
UID Generation Algorithm

Performance

- The firmware code size is within the 8K bytes ROM memory limit of the microcontroller used.
- The firmware code needs atleast 1 second per UID to function correctly.
- With the provision of 8 separate testlines, the total number of UIDs which can be assigned in an 8 hour working day is
  (num_seconds X num_minutes X num_hours X num_testlines) = (60 X 60 X 8 X 8) = 2,30,400
  which seems to be more than adequate for real world scenarios
- Even in the unlikely event of an error occurring in the assignment of the uniqueness part of the UID, the extremely low probability of collision driven by the random section ensures trouble-free UID assignment. (Although, errors in the UID will violate the chronological order and/or supplied criteria values)
- As implemented in the algorithm, the probability of collision between two different UIDs is **$1 / 2^{55} = 2.77 \text{ X } 10^{-17}$**.
- Of course, even in the unlikely event of a random section collision, the uniqueness section of the UID ensures that each UID remains unique atleast within the vendor's premises.

The actual listing of software deliverables, the build process and the UID generation procedure are covered in a separate document. (***Code_Operation.pdf***)
