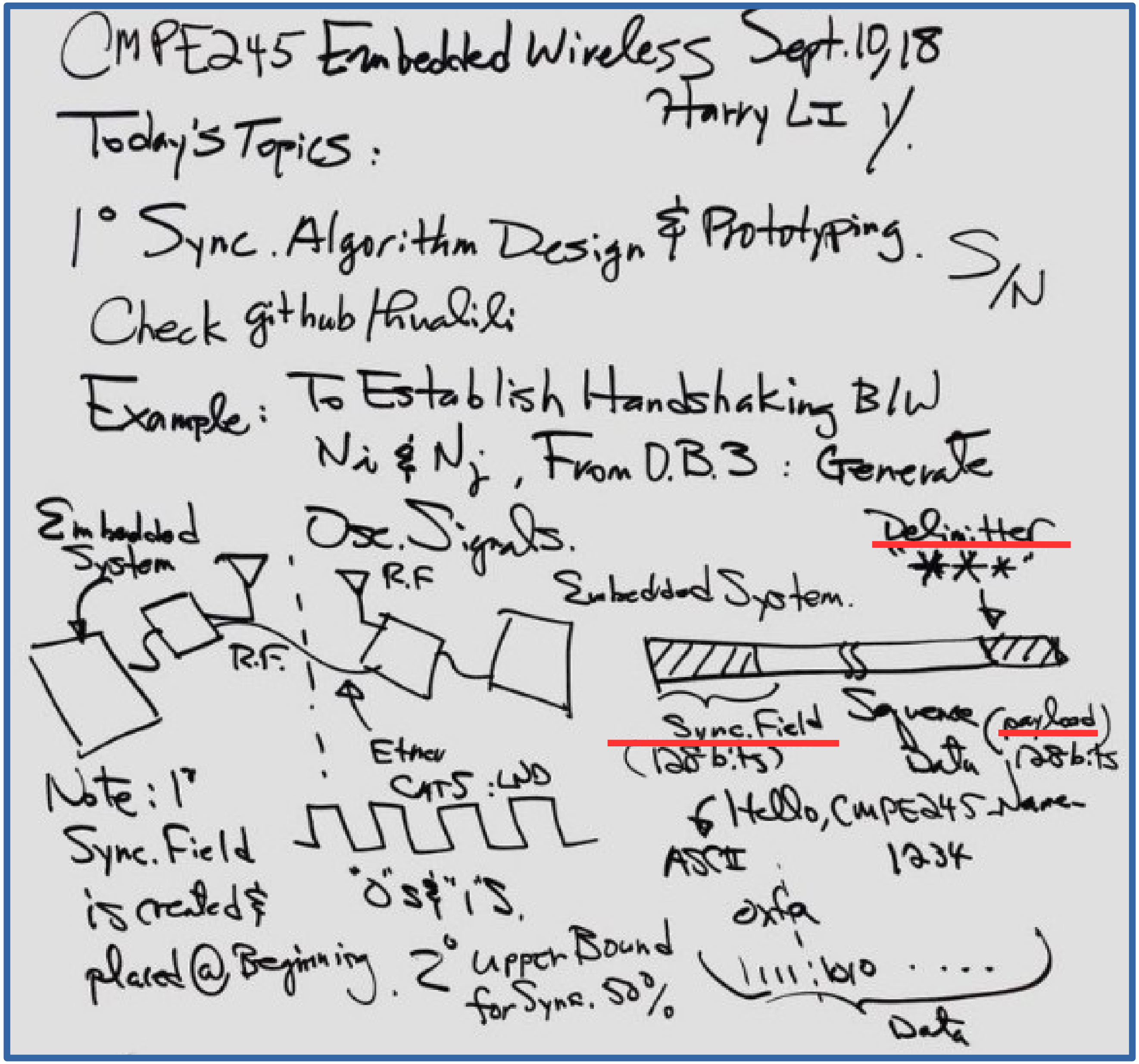
9-10-2018 Sync Design



CMPE245 Embedded Wireless Sept. 10, 18

Harry LI

Today's Topics:

1° Sync. Algorithm Design & Prototyping. S/N

Check github /Hualili

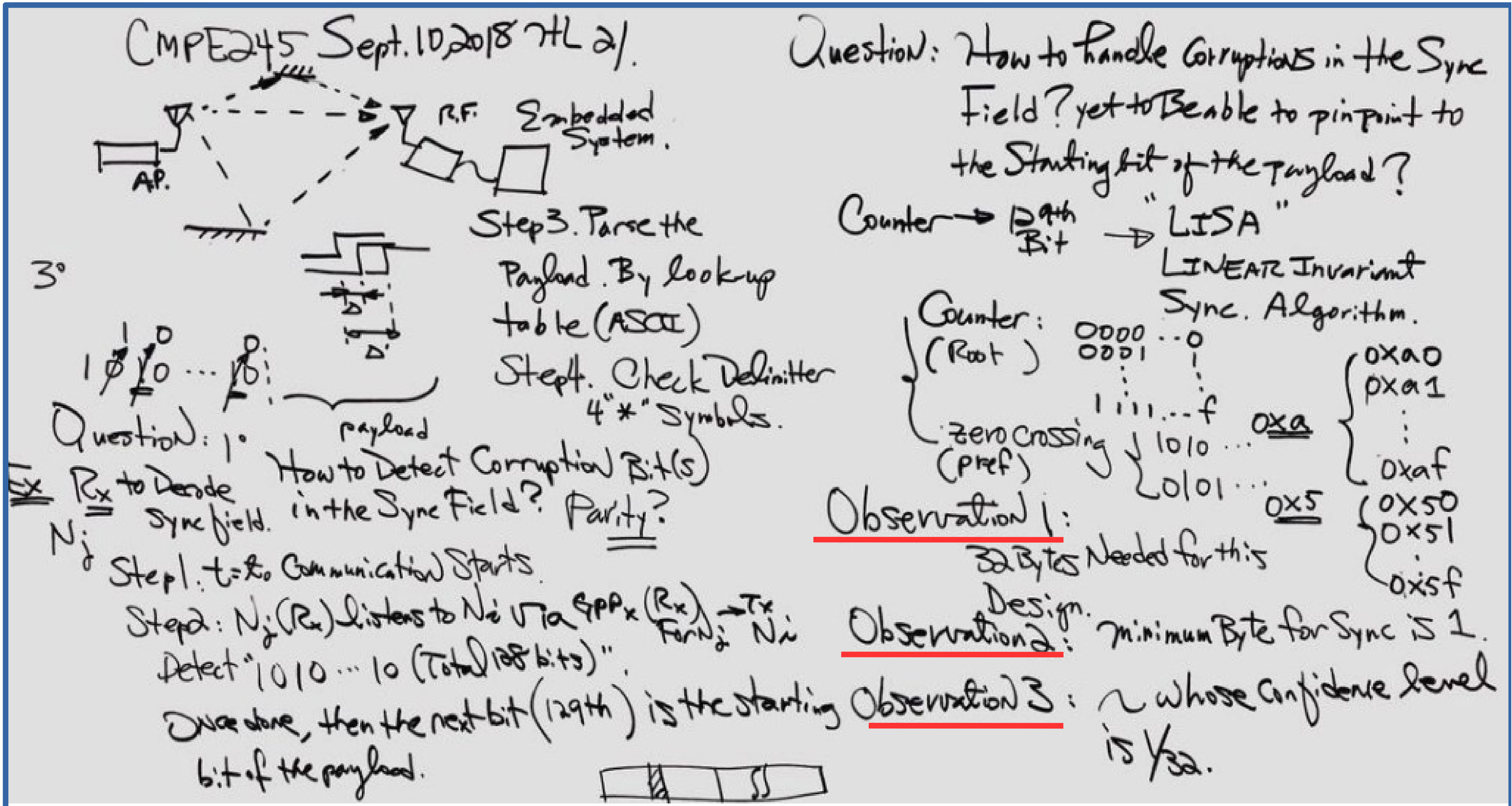
Example: To Establish Handshaking B/W Ni & Nj, From D.B.3: Generate Osc. Signals.

Embedded System
R.F
R.F.
Ether
CAT5 : WD
"0"s & "1"s.
Embedded System.
Delimitter
"***"
Sync. Field
(128 bits)
Sequence
Data
(payload)
128 bits
& Hello, CMPE245 Name
ASCII
1234
0x1a
1111:1010 . . . .
Data

Note: 1° Sync. Field is created & placed @ Beginning. 2° Upper Bound for Sync. 50%

Harry Li, Ph.D.

9-9-2018 LISA Algorithm


R.F.
Embedded System.
AP.
3°
Δ
Δ'
1 0
1 ø 1 0 ... 1 0
0
payload

CMPE245 Sept. 10, 2018 HL 21.

Step 3. Parse the Payload. By look-up table (ASCII)

Step 4. Check Delimitter 4 "*" Symbols.

Question: 1° How to Detect Corruption Bit(s) in the Sync Field? Parity?

Ex Rx to Decode Sync field. N_j

Step 1. t = t_0 Communication Starts.

Step 2: N_j (Rx) listens to N_i via SPP_x (Rx) For N_j → Tx N_i

Detect "1010 ... 10 (Total 128 bits)".

Once done, then the next bit (129th) is the starting bit of the payload.

Question: How to handle Corruptions in the Sync Field? yet to Be able to pinpoint to the Starting bit of the payload?

Counter → 129th Bit → "LISA" LINEAR Invariant Sync. Algorithm.

Counter: (Root) 0000 ... 0, 0001, ..., 1111 ... f

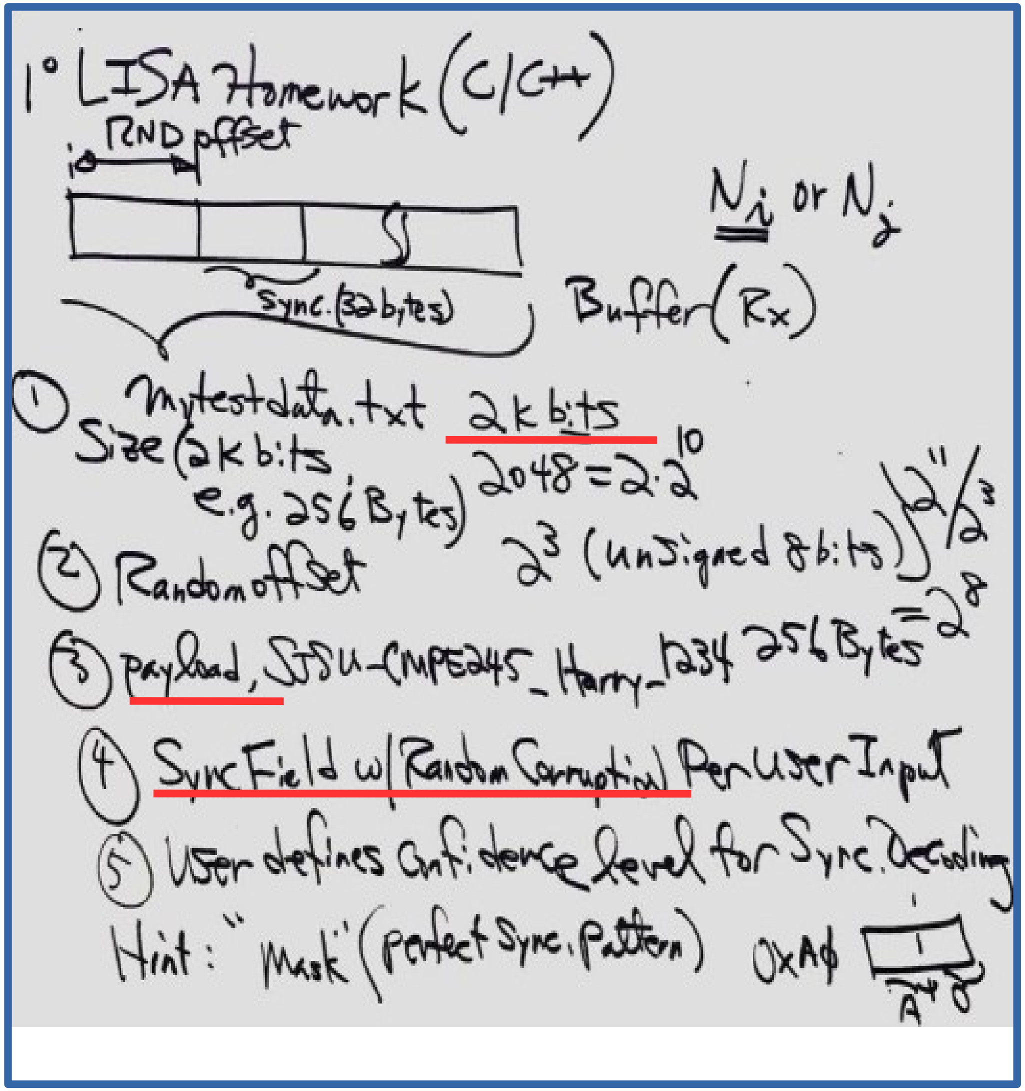
zero crossing (pref) 1010 ... 0xa {0xa0, 0xa1, ..., 0xaf}; 0101 ... 0x5 {0x50, 0x51, ..., 0x5f}

Observation 1: 32 Bytes Needed for this Design.

Observation 2: minimum Byte for Sync is 1.

Observation 3: ~ whose Confidence level is 1/32.

9-12-2018 LISA Implementation



1° LISA Homework (C/C++)

i0 RND offset

N_i or N_j

Sync. (32 bytes)

Buffer (Rx)

① Mytestdata.txt 2k bits
Size (2k bits
e.g. 256 Bytes) $2048 = 2 \cdot 2^{10}$

2^3 (unsigned 8 bits)) $2^{11}/2^3$

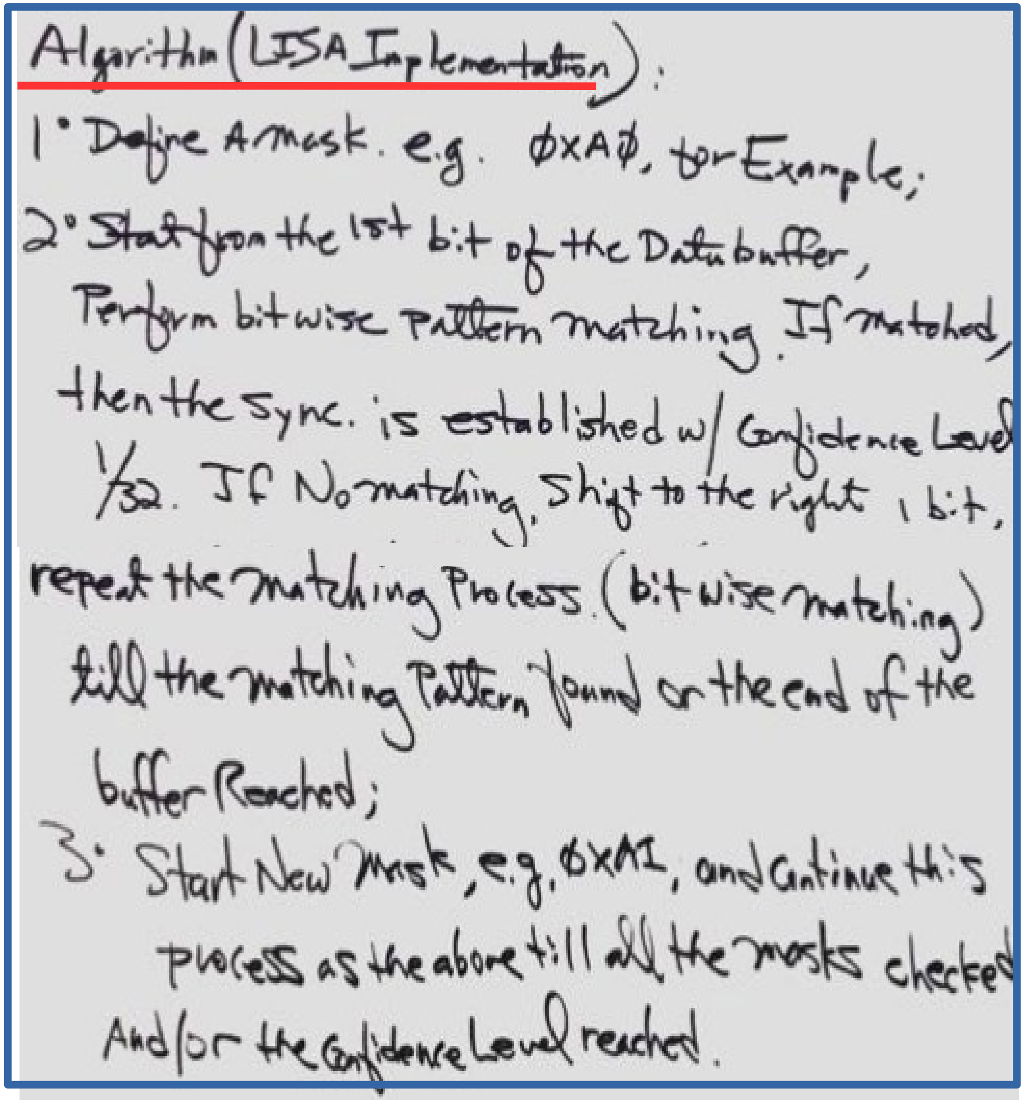
② Randomoffset

③ payload, SJSU-CMPE245_Harry_1234 256 Bytes $= 2^8$

④ SyncField w/ Random Corruption Per User Input

⑤ User defines Confidence level for Sync. Decoding

Hint: "mask" (perfect Sync. pattern) 0xA0 1 A 0



Algorithm (LISA Implementation):

1° Define A mask. e.g. 0xA0, for Example;

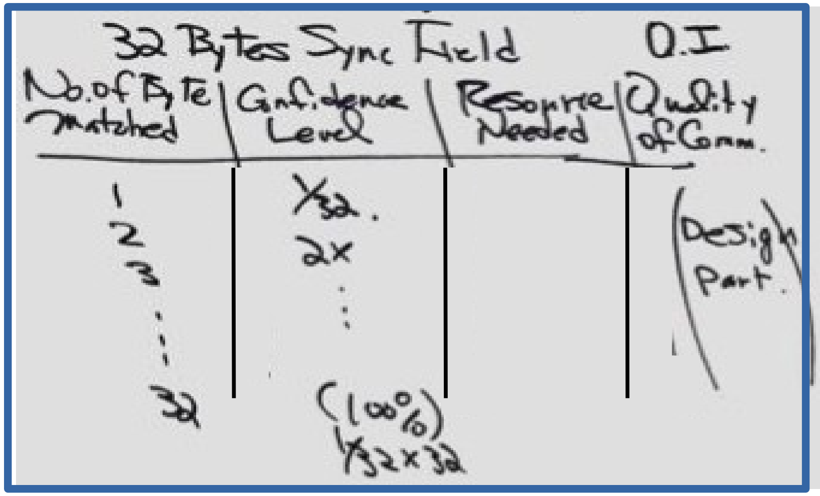
2° Start from the 1st bit of the Data buffer, Perform bitwise pattern matching. If matched, then the Sync. is established w/ Confidence Level 1/32. If No matching, Shift to the right 1 bit, repeat the matching Process (bitwise matching) till the matching Pattern found or the end of the buffer Reached;

3° Start New Mask, e.g. 0xA1, and Continue this process as the above till all the masks checked And/or the Confidence Level reached.

Harry Li, Ph.D.

9-17-2018 LISA C/C++ Implementation

Table 1. Sync Bytes vs. Confidence Level, vs. QI index

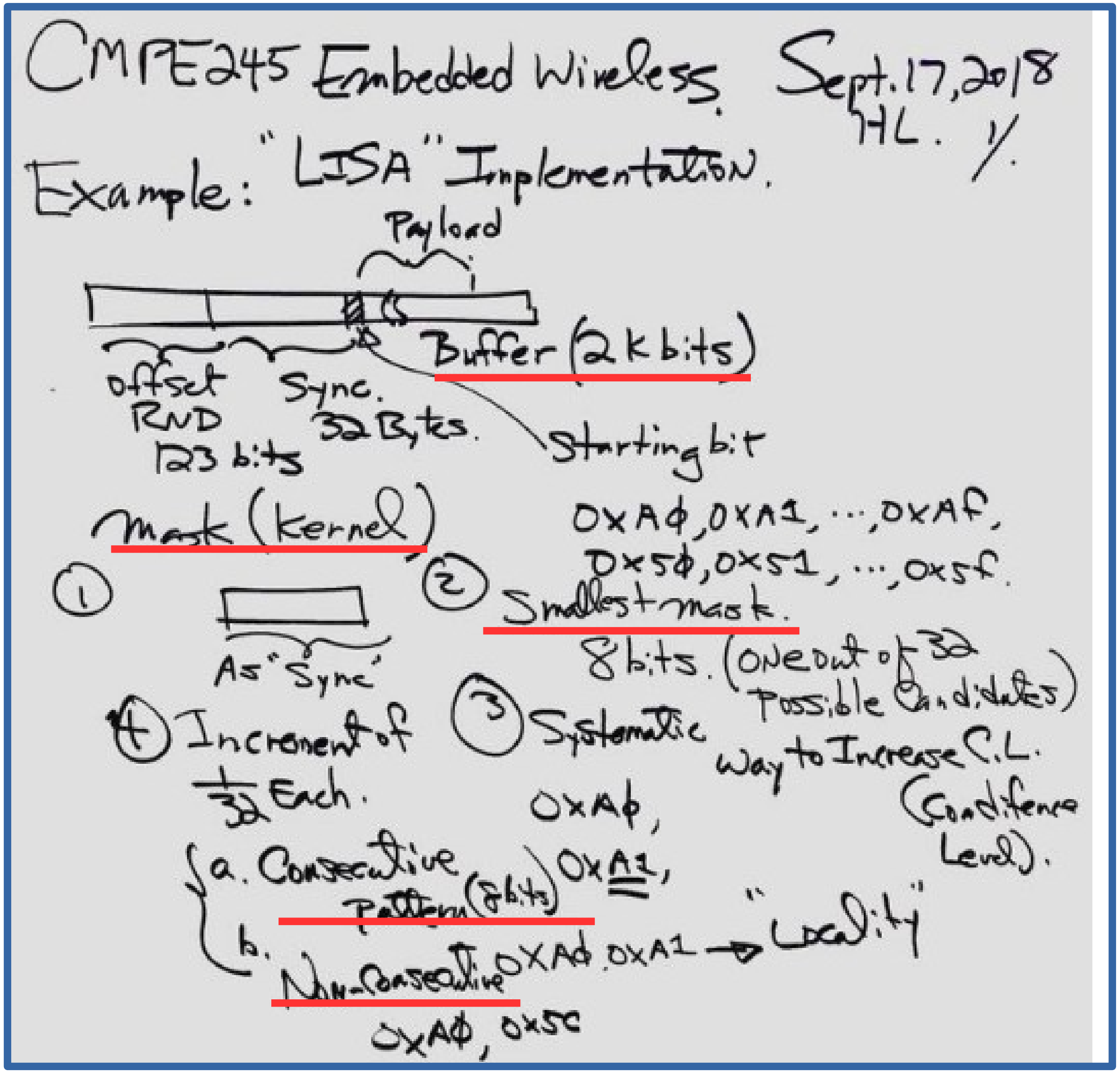


32 Bytes Sync Field Q.I

No. of Byte matched	Confidence Level	Resource Needed	Quality of Comm.
1	$\frac{1}{32}$.		(Design Part.
2	2x		
3	⋮		
⋮			
32	(100%) $\frac{1}{32} \times 32$		)

LISA For Sync Adaptation

Define QI (quality index) between Node i and Node j communication.



CMPE245 Embedded Wireless. Sept. 17, 2018 HL. 1/.

Example: "LISA" Implementation.

Payload

offset RND 123 bits

Sync. 32 Bytes.

Buffer (2k bits)

Starting bit

Mask (kernel)

0xAØ, 0xA1, …, 0xAf,
0x5Ø, 0x51, …, 0x5f.

① As "Sync"

② Smallest mask. 8 bits. (One out of 32 Possible Candidates)

③ Systematic way to Increase C.L. (Condifence Level).

④ Increment of $\frac{1}{32}$ Each.

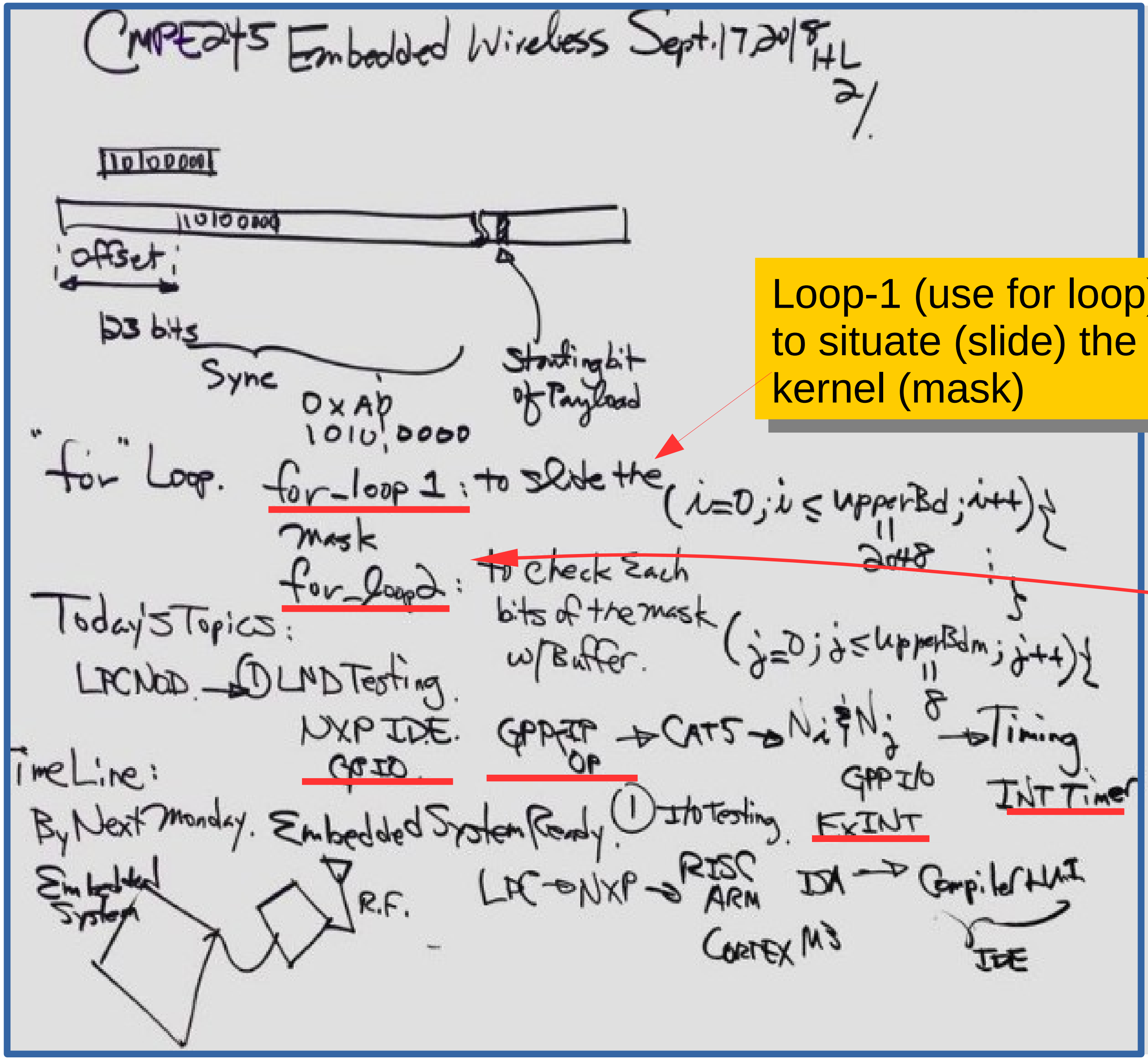
a. Consecutive Pattern (8 bits) 0xAØ, 0xA1,

b. Non-Consecutive 0xAØ, 0xA1 → "Locality"

0xAØ, 0x5C

Harry Li, Ph.D.

9-17-2018 C/C++ Sync Implementation



CMPE245 Embedded Wireless Sept.17,2018 HL

2/.

1010 0000
1010 0000
offset
123 bits
Sync
Starting bit of Payload

0xA0
1010,0000

"for" Loop. for_loop 1: to slide the mask (i=0; i ≤ upperBd; i++){

= 2048

;

}

for_loop2: to check each bits of the mask w/Buffer. (j=0; j ≤ upperBdm; j++){

= 8

Today's Topics:

LPCNOD → (1) LND Testing.

NXP IDE. GPIO. GPIO IP OP → CAT5 → Ni & Nj → Timing GPP I/O INT Timer

TimeLine:

By Next Monday. Embedded System Ready. (1) I/O Testing. ExINT

Embedded System
R.F.

LPC → NXP → RISC ARM Cortex M3 IDA → Compiler HAL IDE

Loop-1 (use for loop) to situate (slide) the kernel (mask)

Loop-2 (use for loop) to check the matching of the kernel (mask) with the data from the buffer

Harry Li, Ph.D.