Canova Tech

The Art of Silicon Sculpting

PIERGIORGIO BERUTO

ANTONIO ORZELLI

IEEE802.3cg TF

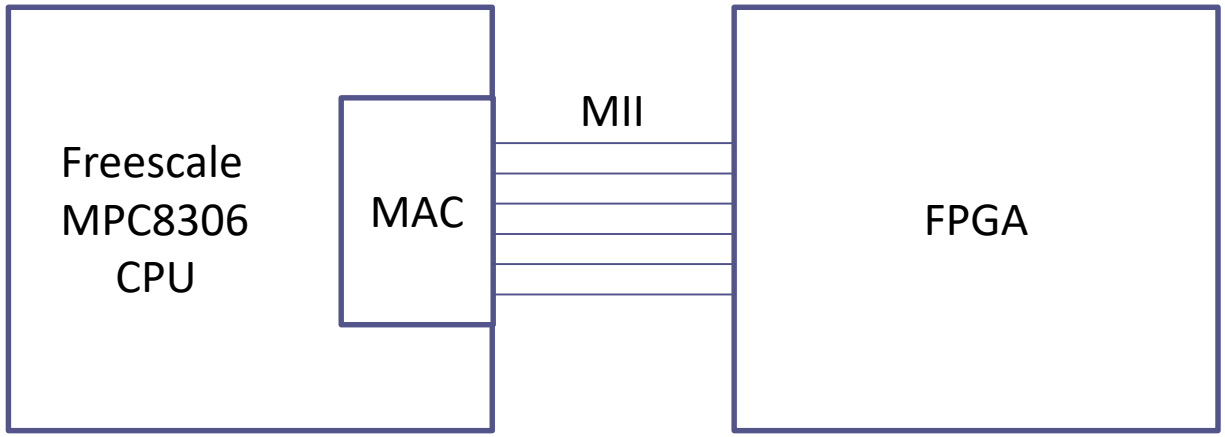
PLCA & Multiple Collisions

April 11th, 2018

Outline

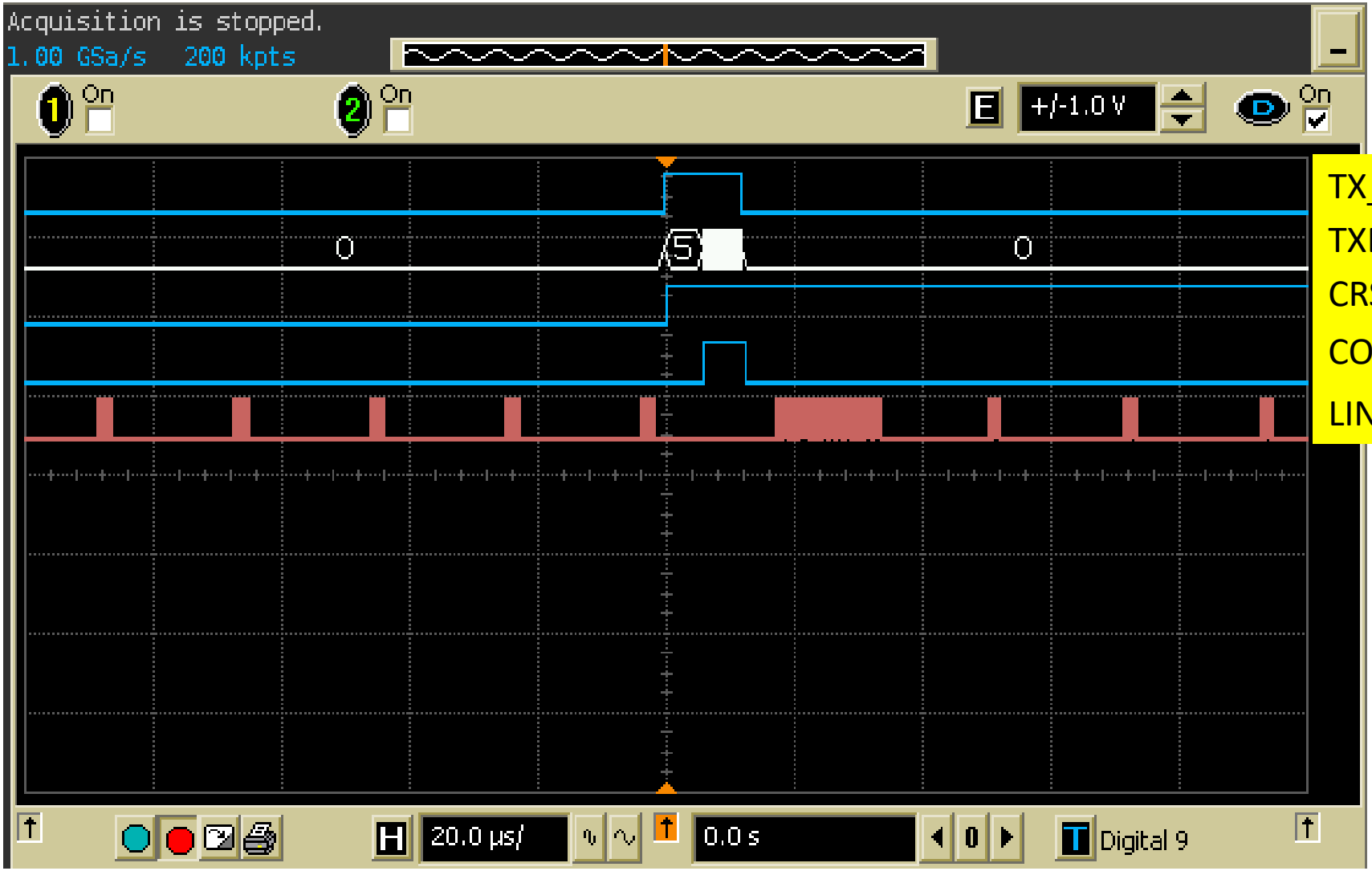
- Some doubts were raised about MAC expected behavior when CRS is high after a collision:
  - According to one interpretation of clause 4, the MAC is allowed to transmit after the back-off period despite the state of CRS
    - This would mean that PLCA would not guarantee bounded latency and fairness due to multiple collisions
  - According to our understanding, the MAC shall wait for CRS de-assertion before making a new transmit attempt
    - PLCA actively relies on this to defer the transmission until the next transmit opportunity is met
      - no multiple collisions are possible with PLCA (max attempts = 1)

Test on real HW

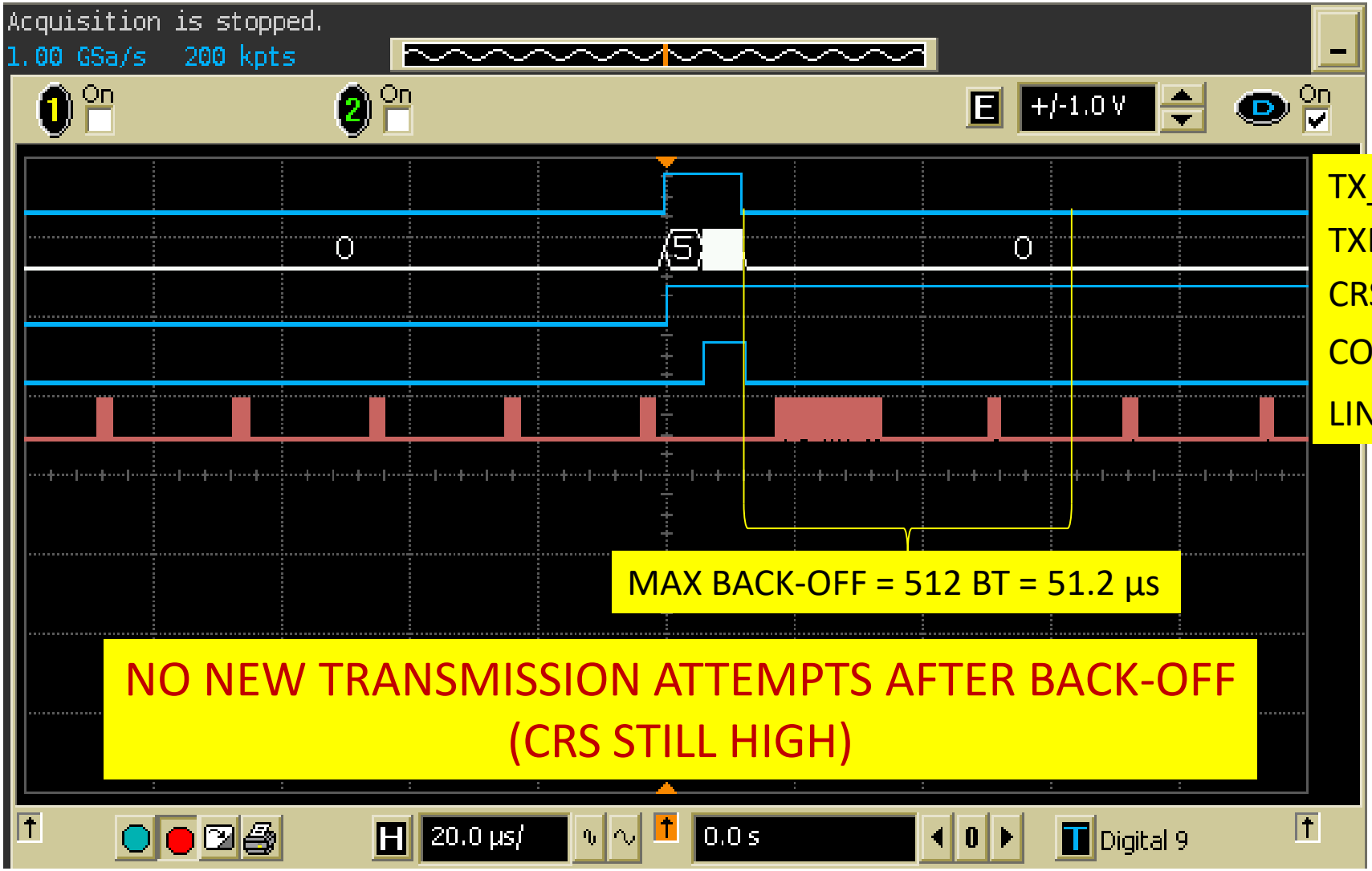


- Embedded MAC
  - Configured for 10 Mbps, Half-Duplex operations
- Dedicated FPGA code
  1. detect TX_EN asserted
  2. wait a few MII clock cycles
  3. assert CRS
  4. wait some more clock cycles
  5. assert COL, keeping CRS asserted
  6. Wait for manual command to de-assert CRS

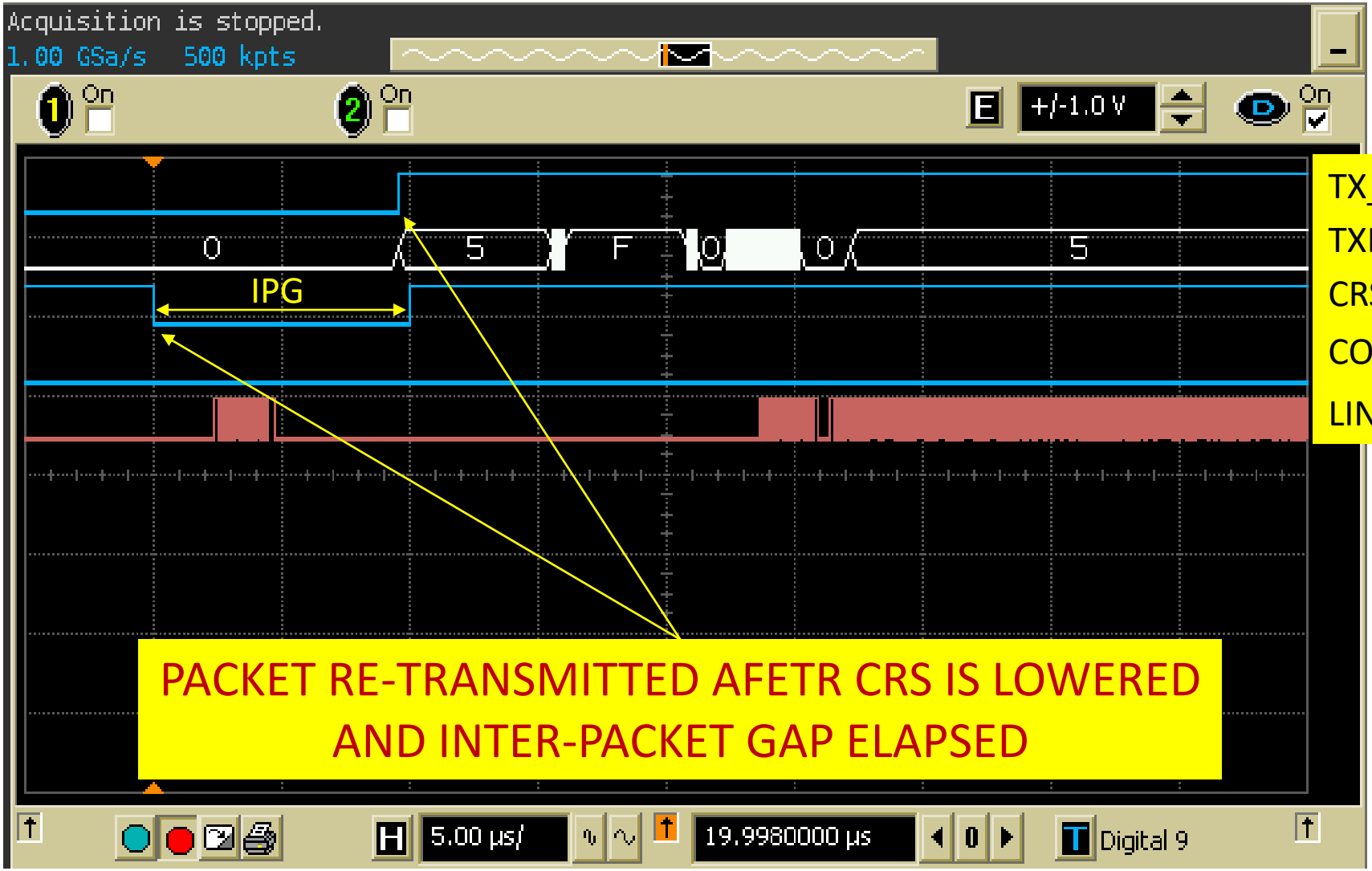
Step 1 - 5


Acquisition is stopped.
1.00 GSa/s 200 kpts
On
On
+/-1.0 V
On
TX_EN
TXD
CRS
COL
LINE
20.0 µs/
0.0 s
Digital 9

Step 1 - 5


Acquisition is stopped.
1.00 GSa/s 200 kpts
On
On
+/-1.0 V
On
TX_EN
TXD
CRS
COL
LINE
MAX BACK-OFF = 512 BT = 51.2 µs
NO NEW TRANSMISSION ATTEMPTS AFTER BACK-OFF
(CRS STILL HIGH)
20.0 µs/
0.0 s
Digital 9

Step 6


Acquisition is stopped.
1.00 GSa/s 500 kpts
1 On
2 On
E +/-1.0 V
D On
0
5
F
0
0
5
IPG
TX_EN
TXD
CRS
COL
LINE
PACKET RE-TRANSMITTED AFETR CRS IS LOWERED
AND INTER-PACKET GAP ELAPSED
H 5.00 µs/
19.9980000 µs
0
T Digital 9

Conclusions

- Multiple collisions with PLCA are not possible
  - Back-off worst case time is 512 BT, which is less than the minimum packet size
- Round-Robin access to the media is guaranteed under any circumstance
  - improves CSMA/CD

Thank You !

FORMAL DEMONSTRATION

Cl 4.2.8: function TransmitLinkMgmt

```
attempts := 0
transmitSucceeding := false

while (attempts < attemptLimit) and (not transmitSucceeding) and (not extend or lateCollisionCount = 0) do
                {No retransmission after late collision if operating at 1000 Mb/s}
begin {Loop}
        if bursting then {This is a burst continuation}
                frameWaiting := true {Start transmission without checking deference}
        else {Non bursting case, or first frame of a burst}
        begin
                if attempts>0 then BackOff;
                frameWaiting := true;
                while deferring do {Defer to passing frame, if any}
                                if halfDuplex then deferred := true;
                burstStart := true;
                if burstMode then bursting := true
        end;

        lateCollisionError := false;
        StartTransmit;
        frameWaiting := false;
        if halfDuplex then
        begin
                while transmitting do WatchForCollision;
                if lateCollisionError then lateCollisionCount := lateCollisionCount + 1;
                attempts := attempts + 1
        end {Half duplex mode}
        else while transmitting do nothing {Full duplex mode}
end; {Loop}
```

- Called to transmit a packet
- Synchronous function

Cl 4.2.8: function TransmitLinkMgmt (simplified)

```
attempts := 0
transmitSucceeding := false

while (attempts < attemptLimit) and (not transmitSucceeding) and (not extend or lateCollisionCount = 0) do
                {No retransmission after late collision if operating at 1000 Mb/s}
begin {Loop}
        if bursting then {This is a burst continuation}
                frameWaiting := true {Start transmission without checking deference}
        else {Non bursting case, or first frame of a burst}
        begin
                if attempts>0 then BackOff;
                frameWaiting := true;
                while deferring do {Defer to passing frame, if any}
                        if halfDuplex then deferred := true;
                burstStart := true;
                if burstMode then bursting := true
        end;

        lateCollisionError := false;
        StartTransmit;
        frameWaiting := false;
        if halfDuplex then
        begin
                while transmitting do WatchForCollision;
                if lateCollisionError then lateCollisionCount := lateCollisionCount + 1;
                attempts := attempts + 1
        end {Half duplex mode}
        else while transmitting do nothing {Full duplex mode}
end; {Loop}
```

[...] Wait (slotTime x Random(0, maxBackOff)); [...]

maxBackOff is initially '2' and increases at each call to BackOff procedure but as we'll show we're going to call this one at most once

random integer between 0 and maxBackOff – 1

not relevant

[...] transmitSucceeding := true;
transmitting := true; [...]

if transmitSucceeding *and* collisionDetect *then*
[...] transmitSucceeding := false; [...]
[...] transmitting := false; [...]

This is set by PLCA (COL)

triggered indirectly by another process

not relevant

For half-duplex 10 Mbit/s:

- attemptLimit = 16
- bursting = False
- extend = False
- halfDuplex = True

Cl 4.2.8: function TransmitLinkMgmt (simplified)

```
attempts := 0
transmitSucceeding := false

while (attempts < attemptLimit) and (not transmitSucceeding)
begin {Loop}
    if attempts>0 then Wait (slotTime x Random(0, maxBackOff));
    frameWaiting := true;
    while deferring do {Defer to passing frame, if any}
        deferred := true;

    transmitSucceeding := true;
    transmitting := true;
    frameWaiting := false;

    while transmitting do
        if transmitSucceeding and collisionDetect then
        begin
            transmitSucceeding := false;
            transmitting := false;
        end

    attempts := attempts + 1
end; {Loop}
```

What about deferring?

Cl 4.2.8: process Deference

```
if halfDuplex then cycle{Half duplex loop}
    while not carrierSense do nothing; {Watch for carrier to appear}
    deferring := true; {Delay start of new transmissions}
    wasTransmitting := transmitting;
    while carrierSense or transmitting do wasTransmitting := wasTransmitting or transmitting;
    if wasTransmitting then Wait(interPacketGapPart1) {Time out first part of interpacket gap}
    else begin
        realTimeCounter := interPacketGapPart1;
        repeat
            while carrierSense do realTimeCounter := interPacketGapPart1;
            Wait(1);
            realTimeCounter := realTimeCounter – 1
        until (realTimeCounter = 0)
    end;
    Wait(interPacketGapPart2); {Time out second part of interpacket gap}
    deferring := false; {Allow new transmissions to proceed}
    while frameWaiting do nothing {Allow waiting transmission, if any}
end {Half duplex loop}
else cycle {Full duplex loop}
            [...]
end; {Deference}
```

- The Deference process runs asynchronously to continuously compute the proper value for the variable *deferring*

Cl 4.2.8: process Deference (simplified)

~~*if* halfDuplex *then cycle*{Half duplex loop}~~

while not carrierSense *do* nothing; {Watch for carrier to appear}

deferring := true; {Delay start of new transmissions}

~~wasTransmitting := transmitting;~~

while carrierSense *or* transmitting *do* ~~wasTransmitting := wasTransmitting *or* transmitting;~~ nothing;

~~*if* wasTransmitting *then* Wait(interPacketGapPart1) {Time out first part of interpacket gap}~~

~~*else begin*~~

~~realTimeCounter := interPacketGapPart1;~~

~~*repeat*~~

~~*while* carrierSense *do* realTimeCounter := interPacketGapPart1;~~

~~Wait(1);~~

~~realTimeCounter := realTimeCounter – 1~~

~~*until* (realTimeCounter = 0)~~

~~*end;*~~

~~Wait(interPacketGapPart2); {Time out second part of interpacket gap}~~

Wait(interPacketGapPart1 + interPacketGapPart2);

deferring := false; {Allow new transmissions to proceed}

while frameWaiting *do* nothing {Allow waiting transmission, if any}

~~*end* {Half duplex loop}~~

~~*else cycle* {Full duplex loop}~~

~~[…]~~

we're half duplex…

~~*end;* {Deference}~~

For 10 Mbit/s, and the sake of this presentation, this simplified to:
Wait(interPacketGapPart1 + interPacketGapPart2)

Cl 4.2.8: process Deference (simplified)

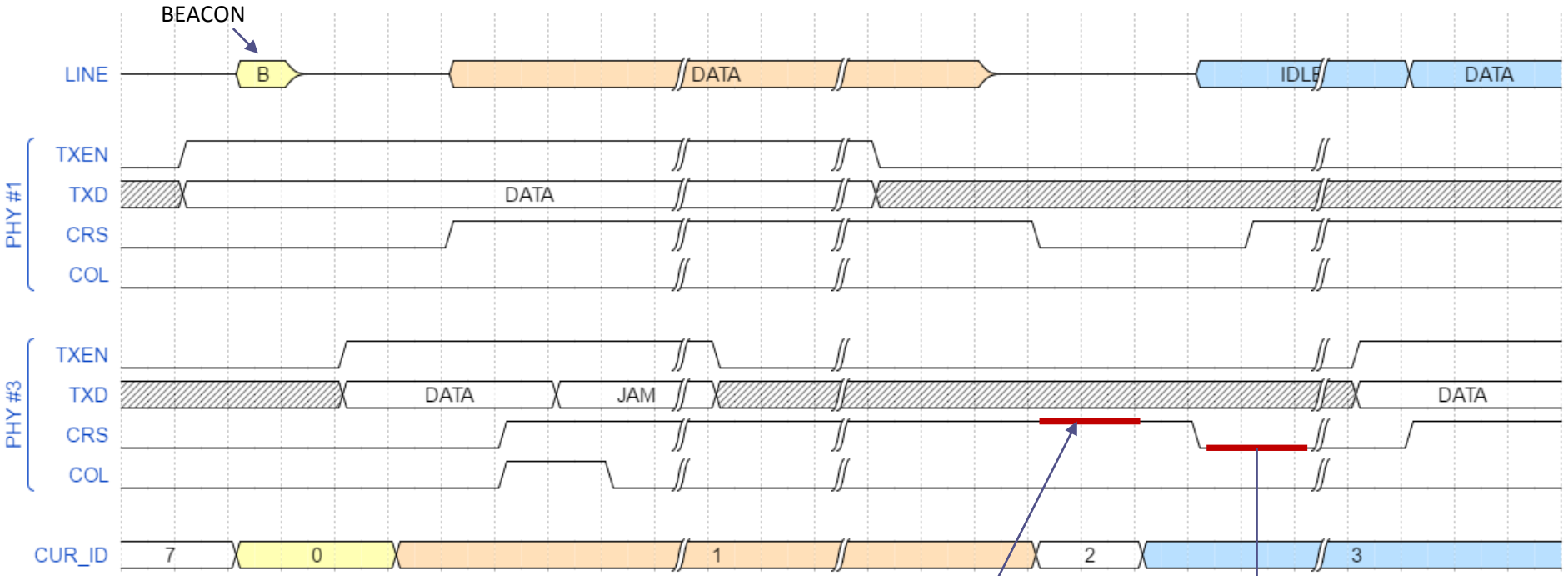
set by PLCA

while not carrierSense *do* nothing; {Watch for carrier to appear}
deferring := true; {Delay start of new transmissions}

while carrierSense *or* transmitting *do* nothing;
Wait(interPacketGapPart1 + interPacketGapPart2);
deferring := false; {Allow new transmissions to proceed}
while frameWaiting *do* nothing; {Allow waiting transmission, if any}

Let's put the two together in case of PLCA collision

Scenario



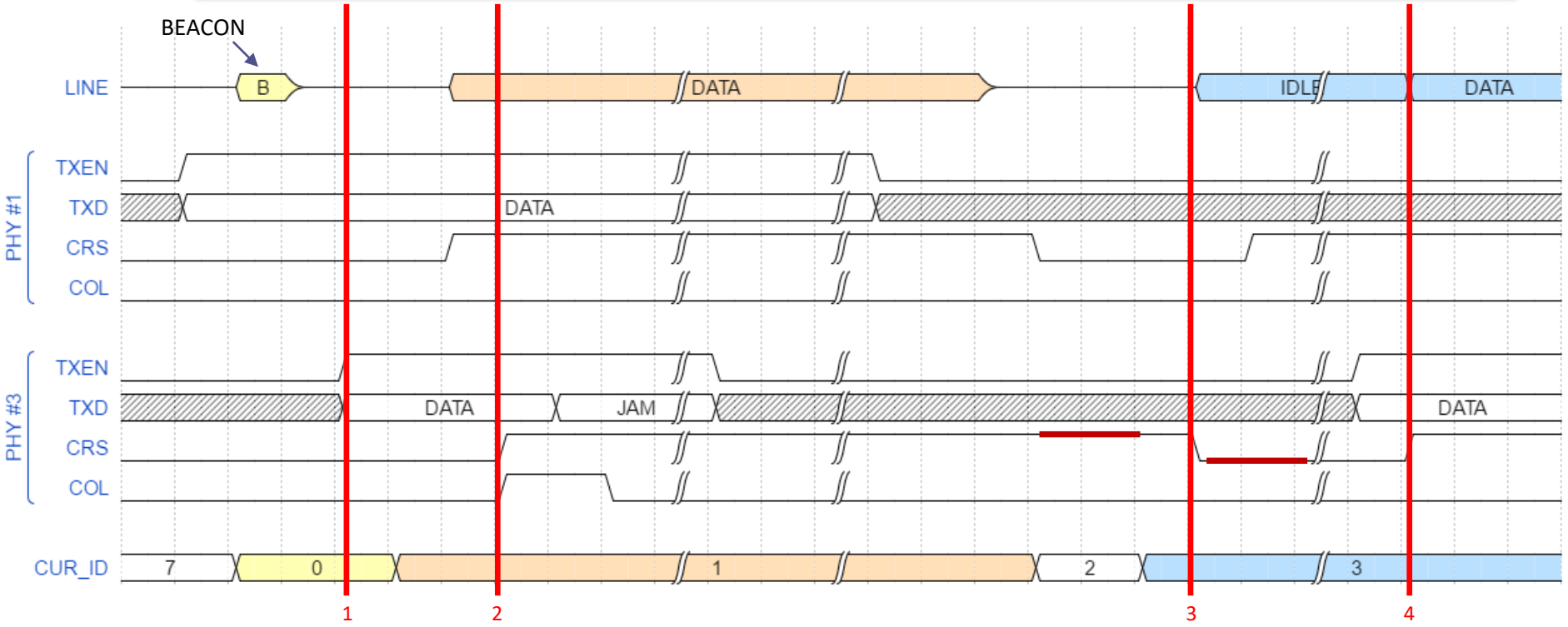
- Node #1 and #3 want to transmit data, others are silent
  - PHY #1 just defers TX until its own transmit opportunity is available
  - PHY #3 signals a collision because PHY #1 is transmitting
  - PHY #3 re-transmits the packet at next transmit opportunity

CRS forced HIGH to prevent the MAC from transmitting until CUR_ID = 3

CRS forced LOW to have the MAC deliver the packet

No multiple collisions - why ?

Scenario (from PHY#3 – MAC#3 perspective)



- Divide the problem in steps, following Pascal code
  1. MAC #3 sends data to PHY #3 to transmit (PLCA starts buffering)
  2. PHY #3 sees carrier and raise a (logical) collision
  3. PHY #3 is allowed by PLCA to re-transmit
  4. PHY #3 receives data from MAC

Step #1

```
attempts := 0
transmitSucceeding := false

while (attempts < attemptLimit) and (not transmitSucceeding)
begin {Loop}
  if attempts>0 then Wait (slotTime x Random(0, maxBackOff));
    frameWaiting := true;
  while deferring do {Defer to passing frame, if any}
        deferred := true;

    transmitSucceeding := true;
    transmitting := true;
    frameWaiting := false;

    while transmitting do
      if transmitSucceeding and collisionDetect then
      begin
            transmitSucceeding := false;
            transmitting := false;
      end

    attempts := attempts + 1
end; {Loop}
```

```
while not carrierSense do nothing; {Watch for carrier to appear}
deferring := true; {Delay start of new transmissions}

while carrierSense or transmitting do nothing;
Wait(interPacketGapPart1 + interPacketGapPart2);
deferring := false; {Allow new transmissions to proceed}
while frameWaiting do nothing {Allow waiting transmission, if any}
```

- MAC #3 is transmitting data (PLCA is buffering)

VARIABLE	VAL
attempts	0
deferring	false
transmitSucceeding	false
transmitting	false
frameWaiting	false

VARIABLE	VAL
attempts	0
deferring	false
transmitSucceeding	true
transmitting	true
frameWaiting	false

Step #2

```
attempts := 0
transmitSucceeding := false

while (attempts < attemptLimit) and (not transmitSucceeding)
begin {Loop}
    if attempts>0 then Wait (slotTime x Random(0, maxBackOff));
    frameWaiting := true;
    while deferring do {Defer to passing frame, if any}
        deferred := true;

    transmitSucceeding := true;
    transmitting := true;
    frameWaiting := false;

    while transmitting do
        if transmitSucceeding and collisionDetect then
        begin
            transmitSucceeding := false;
            transmitting := false;
        end

    attempts := attempts + 1
end; {Loop}
```

```
while not carrierSense do nothing; {Watch for carrier to appear}
deferring := true; {Delay start of new transmissions}

while carrierSense or transmitting do nothing;
Wait(interPacketGapPart1 + interPacketGapPart2);
deferring := false; {Allow new transmissions to proceed}
while frameWaiting do nothing {Allow waiting transmission, if any}
```

- PLCA signals a collision
- TransmitLinkMgmt breaks transmitting loop after jam
- TransmitLinkMgmt perform back-off (0 or 1 slot)
- TransmitLinkMgmt waits “deferring”
- Deference Process waits for carrierSense de-assertion

VARIABLE	VAL
attempts	0
deferring	false
transmitSucceeding	true
transmitting	true
frameWaiting	false

VARIABLE	VAL
attempts	1
deferring	true
transmitSucceeding	false
transmitting	false
frameWaiting	true

Step #3

```
attempts := 0
transmitSucceeding := false

while (attempts < attemptLimit) and (not transmitSucceeding)
begin {Loop}
    if attempts>0 then Wait (slotTime x Random(0, maxBackOff));
    frameWaiting := true;
    while deferring do {Defer to passing frame, if any}
        deferred := true;

    transmitSucceeding := true;
    transmitting := true;
    frameWaiting := false;

    while transmitting do
        if transmitSucceeding and collisionDetect then
        begin
            transmitSucceeding := false;
            transmitting := false;
        end

    attempts := attempts + 1
end; {Loop}
```

```
while not carrierSense do nothing; {Watch for carrier to appear}
deferring := true; {Delay start of new transmissions}

while carrierSense or transmitting do nothing;
Wait(interPacketGapPart1 + interPacketGapPart2);
deferring := false; {Allow new transmissions to proceed}
while frameWaiting do nothing {Allow waiting transmission, if any}
```

- PLCA meets a transmit opportunity
  - carrierSense is de-asserted
- TransmitLinkMgmt still deferring
- Deference Process waits for IPG
- PLCA sends COMMIT to halt CUR_ID progress

VARIABLE	VAL
attempts	1
deferring	true
transmitSucceeding	false
transmitting	false
frameWaiting	true

VARIABLE	VAL
attempts	1
deferring	true
transmitSucceeding	false
transmitting	false
frameWaiting	true

Step #4

```
attempts := 0
transmitSucceeding := false

while (attempts < attemptLimit) and (not transmitSucceeding)
begin {Loop}
    if attempts>0 then Wait (slotTime x Random(0, maxBackOff));
    frameWaiting := true;
    while deferring do {Defer to passing frame, if any}
        deferred := true;

    transmitSucceeding := true;
    transmitting := true;
    frameWaiting := false;

    while transmitting do
        if transmitSucceeding and collisionDetect then
        begin
            transmitSucceeding := false;
            transmitting := false;
        end

    attempts := attempts + 1
end; {Loop}
```

```
while not carrierSense do nothing; {Watch for carrier to appear}
deferring := true; {Delay start of new transmissions}

while carrierSense or transmitting do nothing;
Wait(interPacketGapPart1 + interPacketGapPart2);
deferring := false; {Allow new transmissions to proceed}
while frameWaiting do nothing {Allow waiting transmission, if any}
```

- MAC delivers data to the PHY
- No collisions are possible since CUR_ID == 3
  - packet is sent successfully!

VARIABLE	VAL
attempts	1
deferring	true
transmitSucceeding	false
transmitting	false
frameWaiting	true

VARIABLE	VAL
attempts	1
deferring	false
transmitSucceeding	true
transmitting	true
frameWaiting	false

End of transmission

```
attempts := 0
transmitSucceeding := false

while (attempts < attemptLimit) and (not transmitSucceeding)
begin {Loop}
    if attempts>0 then Wait (slotTime x Random(0, maxBackOff));
    frameWaiting := true;
    while deferring do {Defer to passing frame, if any}
        deferred := true;

    transmitSucceeding := true;
    transmitting := true;
    frameWaiting := false;

    while transmitting do
        if transmitSucceeding and collisionDetect then
        begin
            transmitSucceeding := false;
            transmitting := false;
        end

    attempts := attempts + 1
end; {Loop}
```

```
while not carrierSense do nothing; {Watch for carrier to appear}
deferring := true; {Delay start of new transmissions}

while carrierSense or transmitting do nothing;
Wait(interPacketGapPart1 + interPacketGapPart2);
deferring := false; {Allow new transmissions to proceed}
while frameWaiting do nothing {Allow waiting transmission, if any}
```

- Packet is sent
- Deference Process
  - sees carrierSense due to own transmission
  - waits IPG
  - starts over
- When TransmitLinkMgmt is invoked again
  - attempts is reset (!!)

VARIABLE	VAL
attempts	1
deferring	false
transmitSucceeding	true
transmitting	true
frameWaiting	false

VARIABLE	VAL
attempts	0
deferring	false
transmitSucceeding	false
transmitting	false
frameWaiting	false