Method of handling Burst Start offset and Burst End Offset In EPoC Upstream

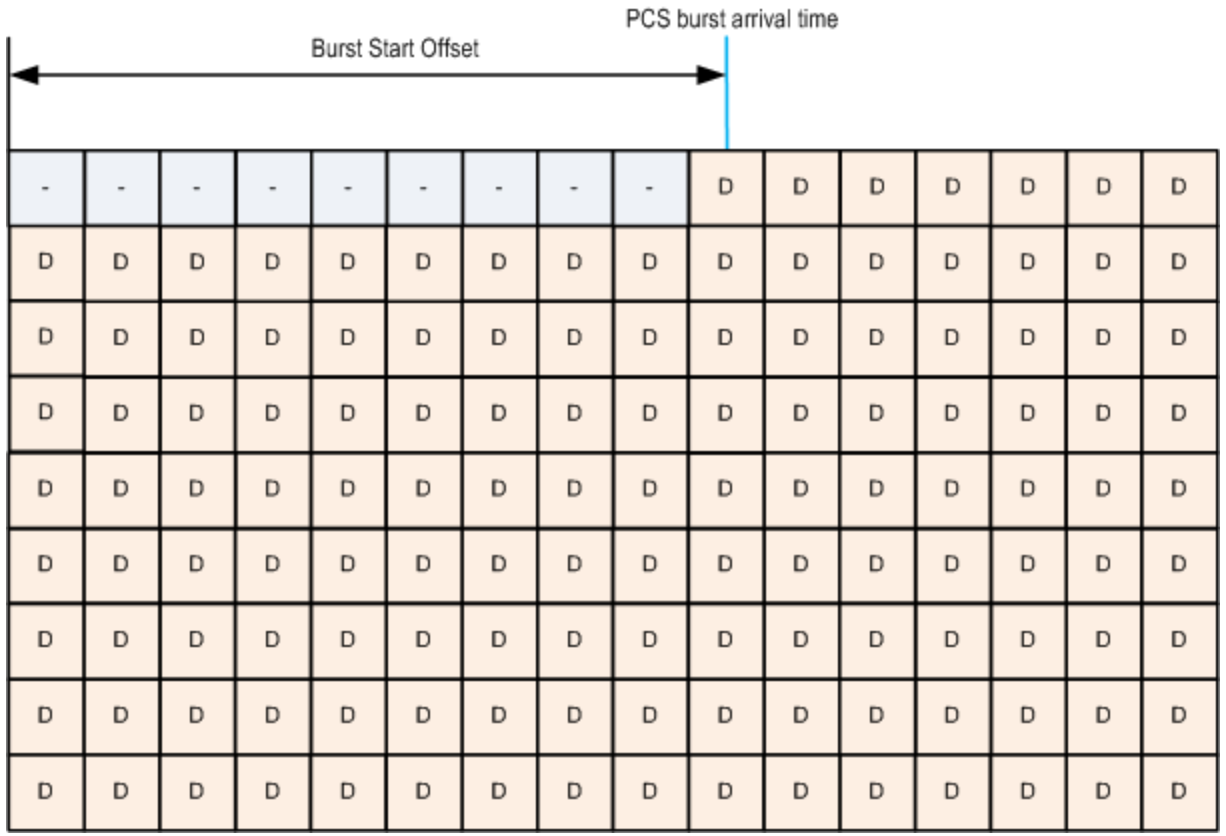
Xiaolin Che, Marvell

Jin Zhang, Marvell

Why does Burst Start Offset Matter?

- MPCP specifies strict jitter requirement
  - Upstream round trip jitter 12TQ, or 192ns, mostly caused by implementation.
  - A jitter free standard is desired
- In zhang_3bn_03a_0714.pdf we show it is feasible to achieve near-free jitter by enforcing a one-to-one mapping between PCS output and 2-D superframe.
- The start marker can only bear information about the position of resource block
- The receiver cannot recover the exact bit location of a burst, causing jitter of maximum 160 bits
  - 1TQ for 10Gbps, not a problem
  - 10TQ for 1Gbps, some problem, need update on MPCP requirement
  - 40TQ for 250Mbps, really a problem, need huge update on MPCP requirement
- A simple solution to get rid of the hassle, no additional signal detection is needed.

Burst Start Offset

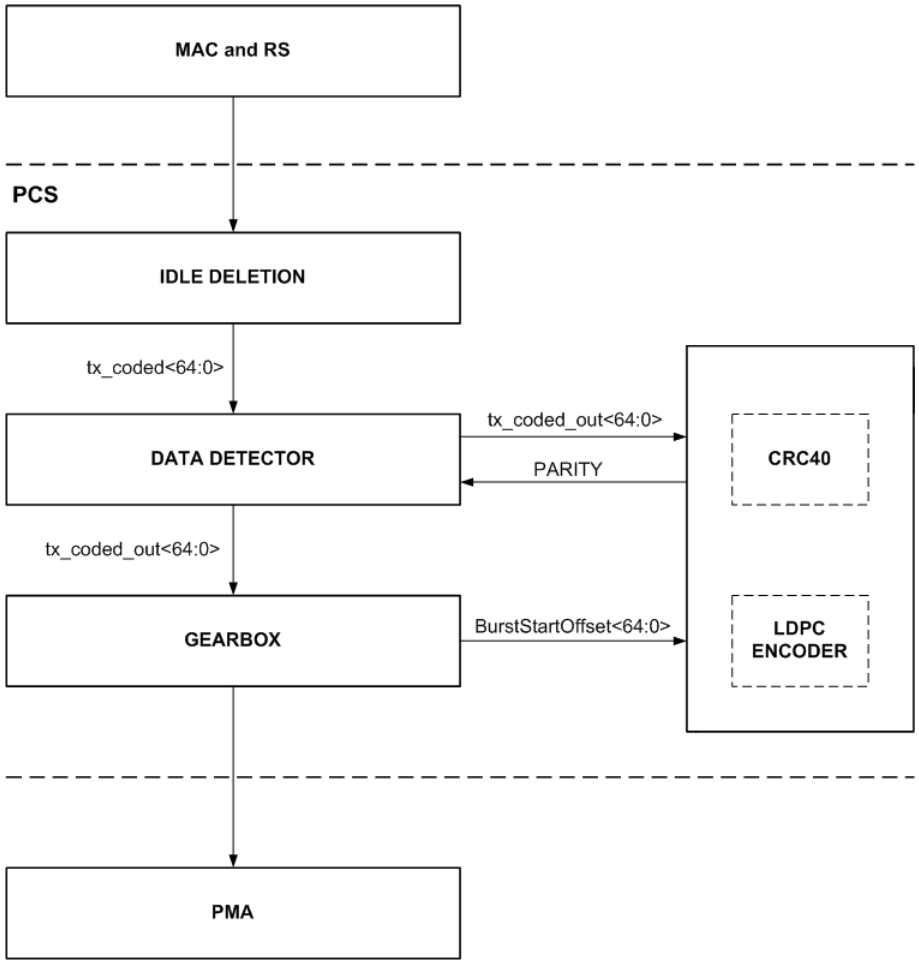


Solution to Burst Start Offset

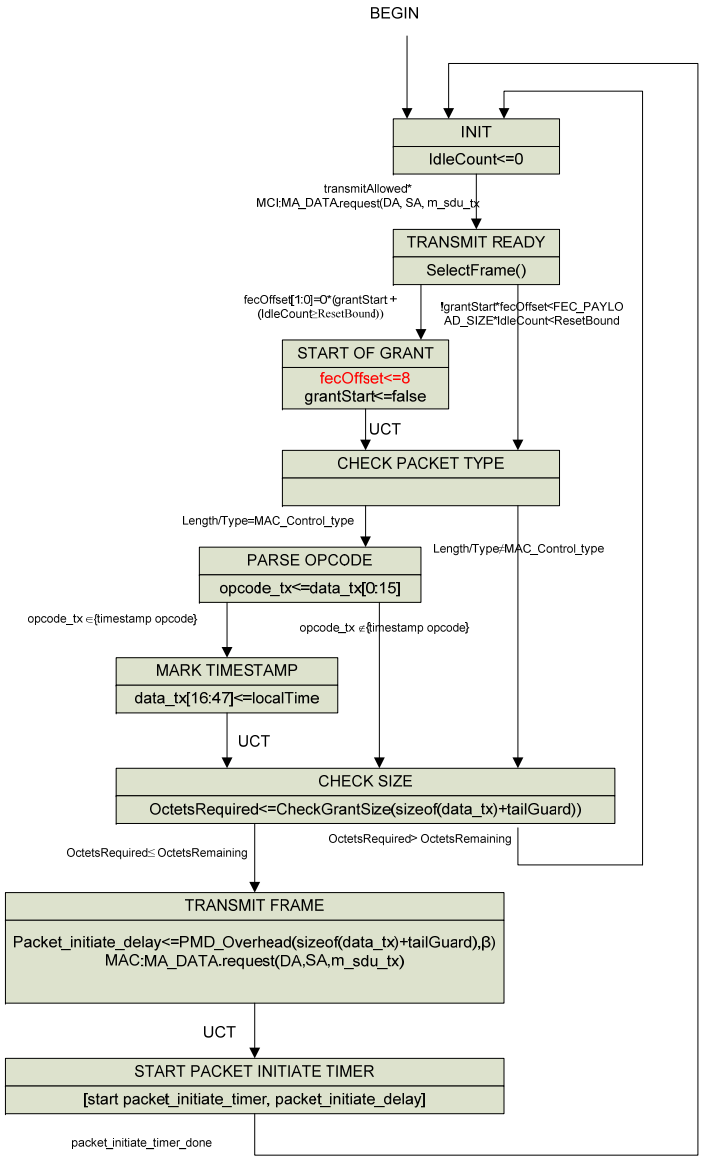
- The PMA burst starts with the first available RE of the first RB of the burst, so no waste of RE at the burst start.
- Send an additional Idle block before the first S block of a burst.
- Upon receiving DataDetected, the Gearbox sends to data detector an 8-bit value of BurstStartOffset.
  - 8 bits are enough to represent the BurstStartOffset as its value is less than 160
- Transmit the BurstStartOffset in the first Idle block of the burst
  - The first bit of the 65-bit Block is sync head. The 8bit BurstStartOffset can be repeated eight times and filled into other 64 bits of the 65-bit block
  - The first block in a burst can be called Burst Start Offset Indicator (BSOI) block

Burst Start Offset Indicator Block

Solution to Burst Start Offset



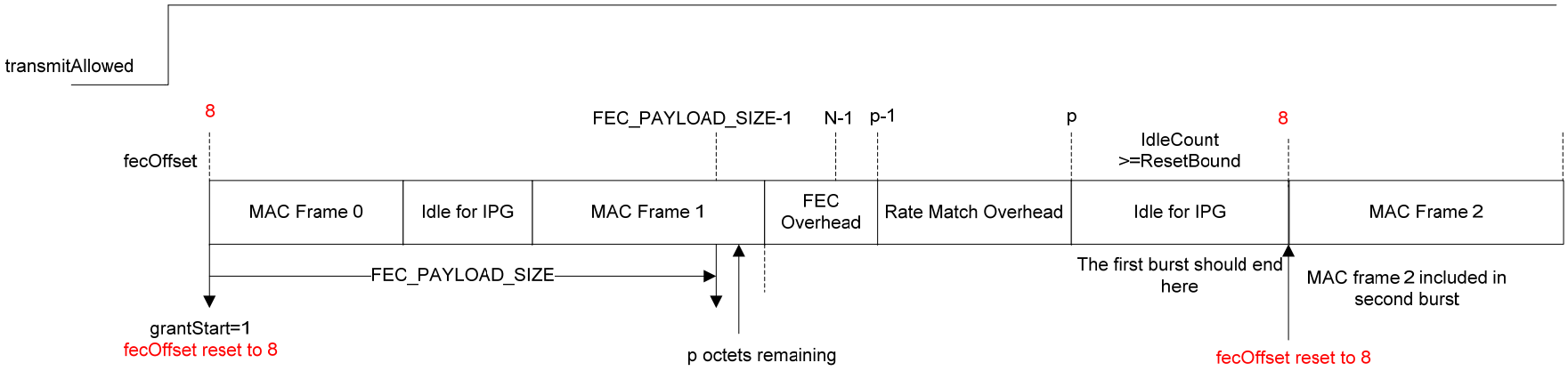
Impact on Control Multiplexer



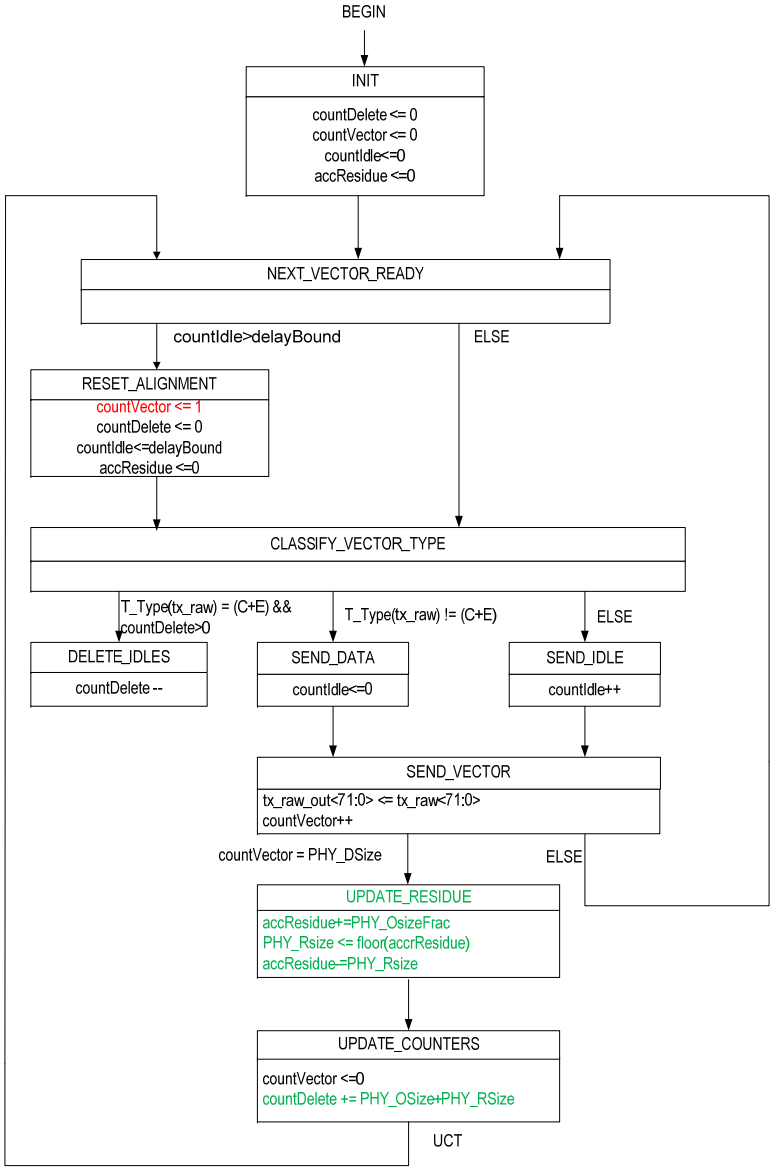
Note:

- FEC_PAYLOAD_SIZE0: payload size in octets of the longest FEC code;
- FEC_CODE_SIZE0:codeword size in octets of the longest FEC code;
- fecOffset shall count from 0 to FEC_CODE_SIZE0+Rate_Match_Overhead-1;
- CheckGrantSize() function shall ensure that after transmitting the incoming frame, the remaining space in the grant is still enough to transmit the remaining fecOffset octets.

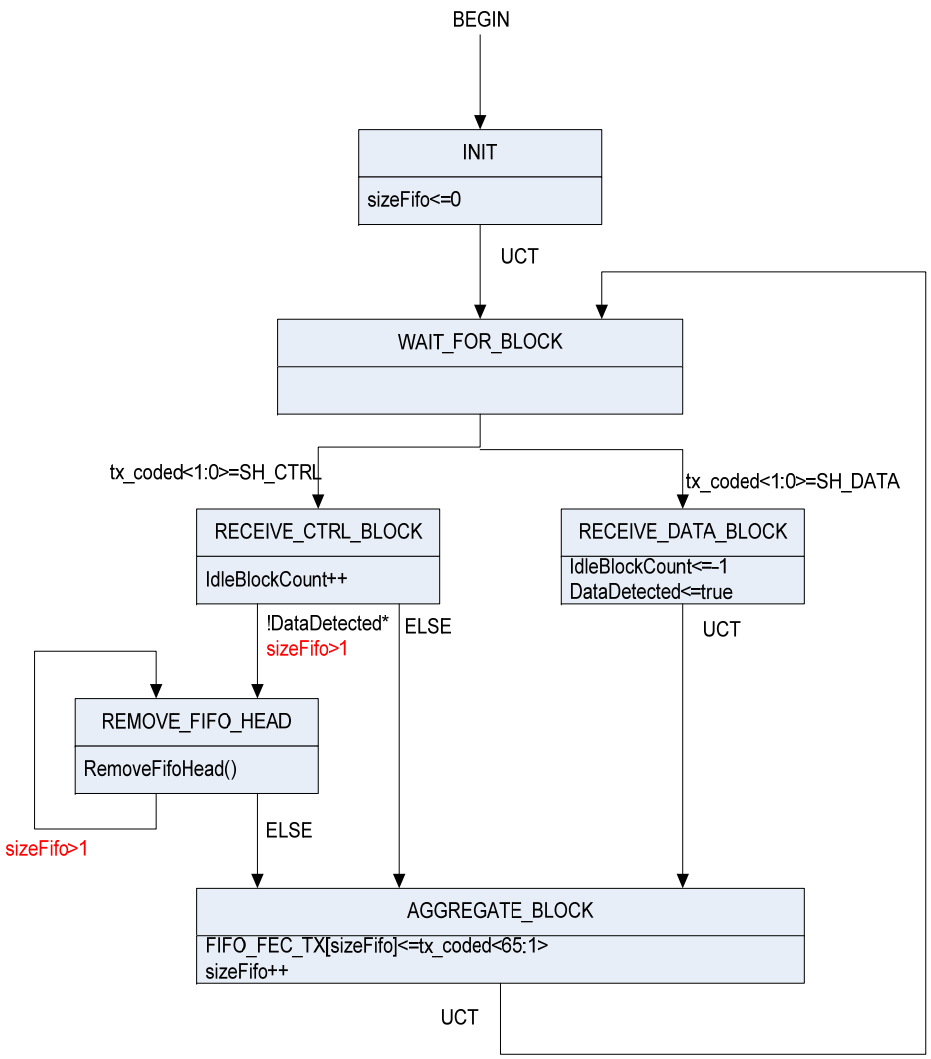
Impact On Control Multilexer



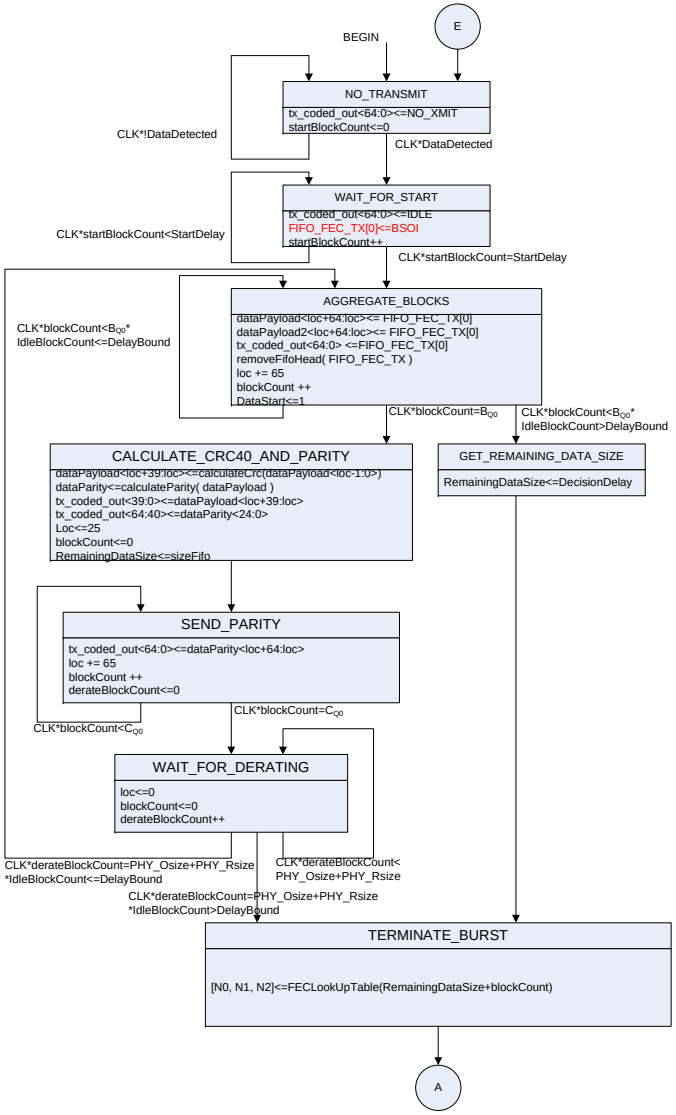
Impact on Idle Deletion



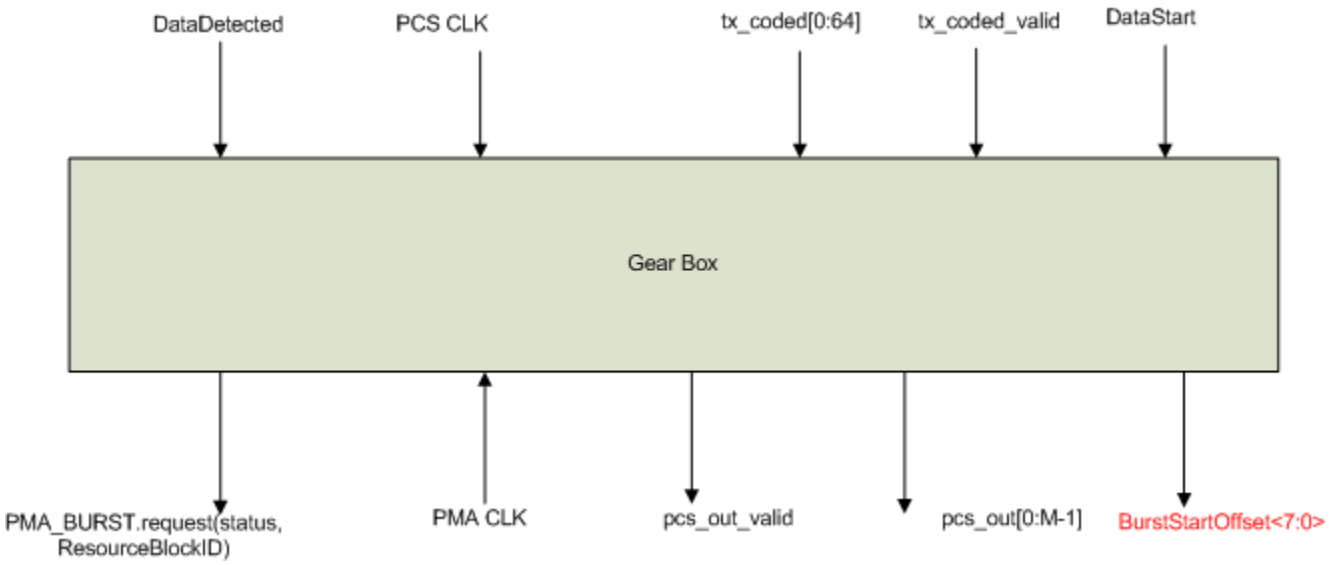
Impact On Data Detector Input Process



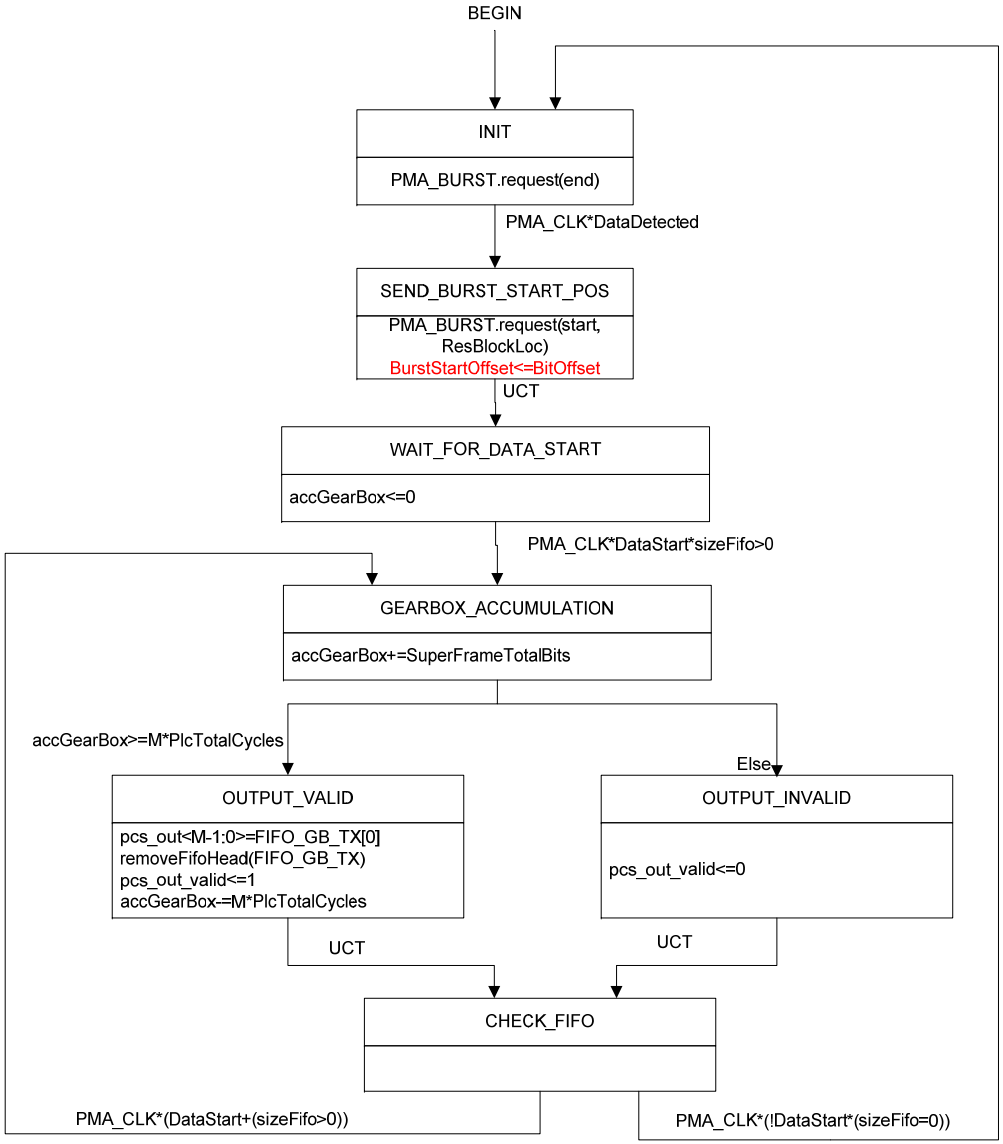
Impact On Data Detector Output Process



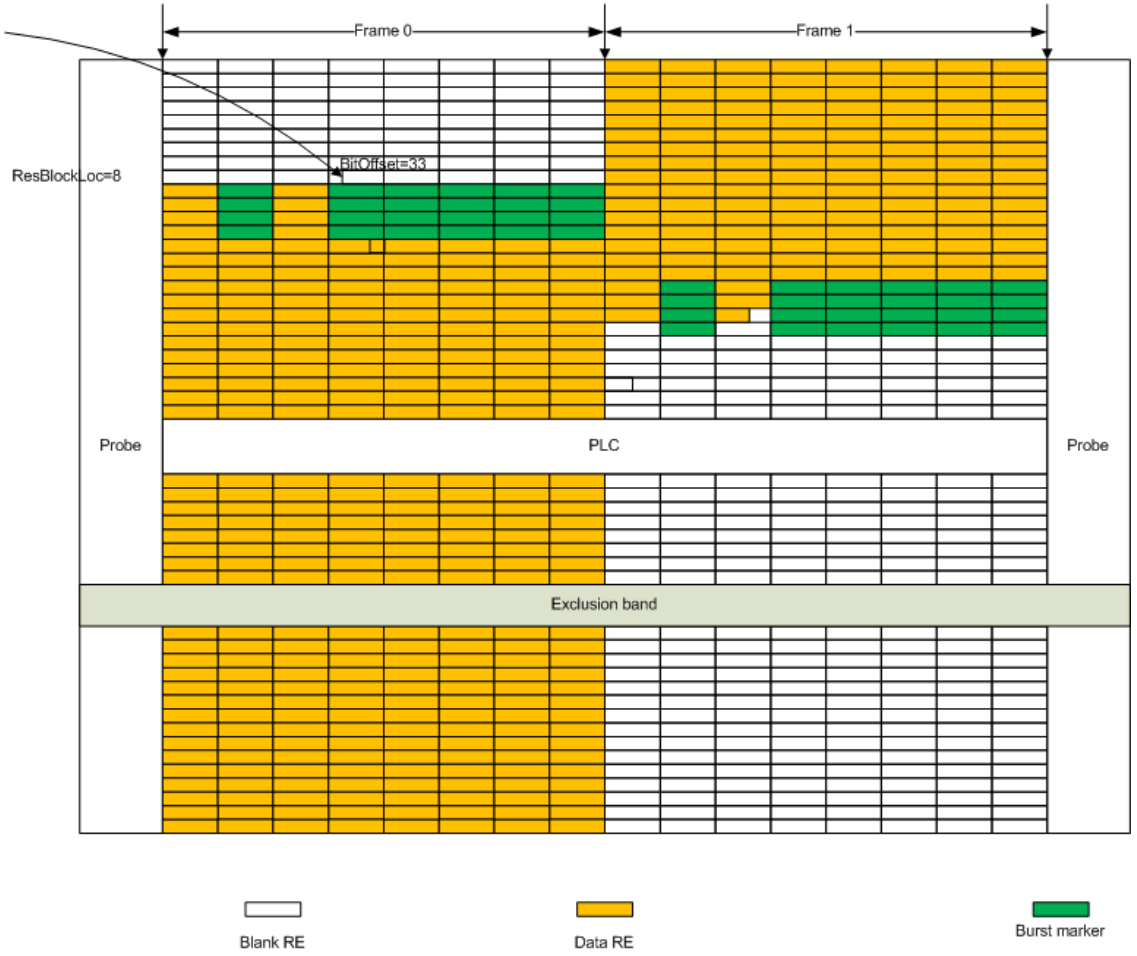
Impact On Gearbox



Impact on Gearbox Output Process



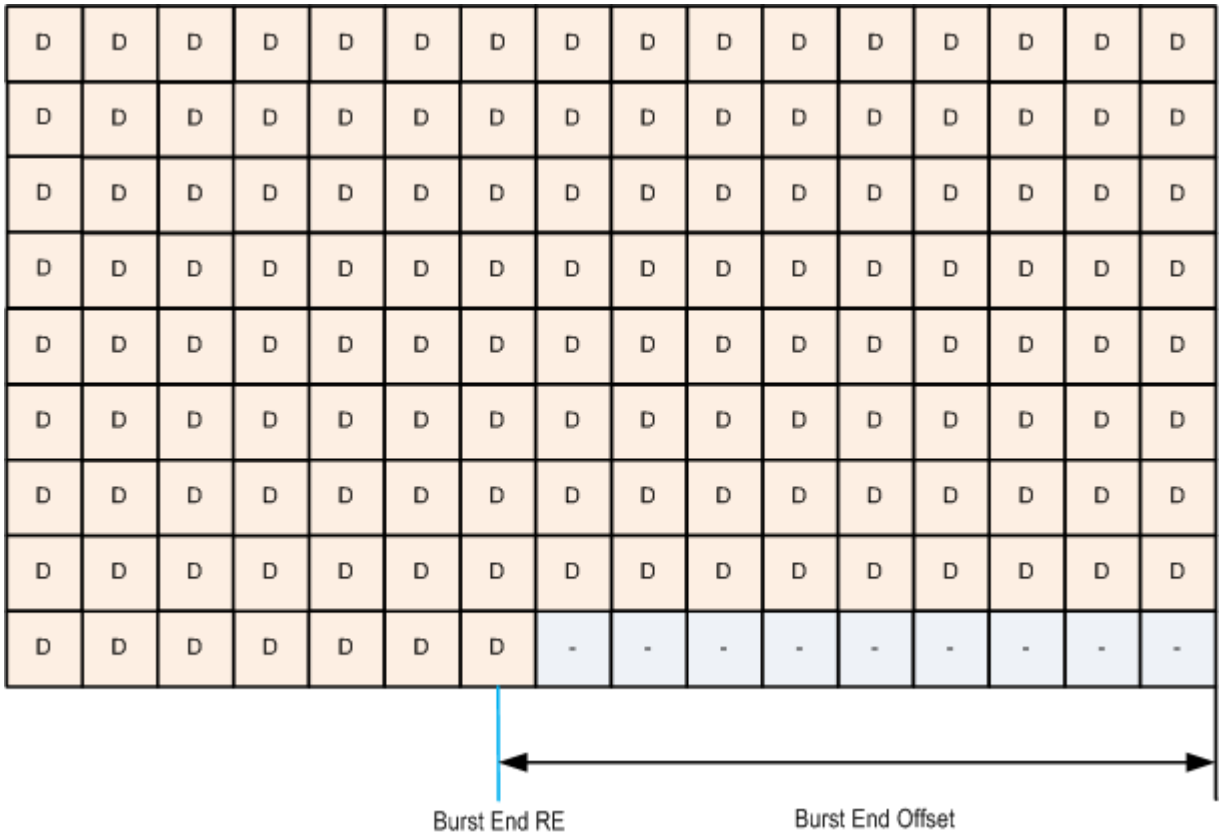
No Waste of RE On the Burst Start Mapping



Burst End Offset

- The upstream burst may not fill up to the last bit of last available RE.
- There is an offset between the position of the last bit of burst data and the last bit of very end, a.k.a Burst End Offset
- The receiver need know the Burst End Offset to obtain the codeword size information.
  - The maximum Burst End Offset might occur in sixteen-OFDM-symbol RB
  - MaxBurstEndOffset=160bits

Burst End offset



Solution to Burst End offset

- Goal:
  - No extra signal detection
  - Minimum cost of efficiency
- Pad IDLEs at the end of the burst so that the granularity of burst size is larger than MaxBurstEndOffset. Then there is no ambiguity for the receiver to locate the burst end RE
- 65*3=195, larger than 160. At most two 65bit IDLE Blocks is required to pad.
- Assuming that burst size is random. 0, 1 or 2 65bit IDLE blocks might need to pad. 0*(1/3) +1*(1/3)+2*(1/3) = 1. The average cost is 65 bits
- The padding Idle blocks shall not change the number of codewords for each FEC size.

Computation of the number of IDLE Blocks to be padded considering codeword filling

- Aside from the full long code, **ProtectedBlockSize** indicates the number of remaining 65bit blocks to be encoded. ProtectedBlockSize<220
- **PadIdleLength** represents the number of the IDLE Blocks required to be padded. It could be computed by the function

PadIdleLength =ComputePadIdleLength(ProtectedBlockSize)

Computation of the number of IDLE Blocks to be padded considering codeword filling

– ComputePadIdleLength(ProtectedBlockSize)

```
ComputePadIdleLength(ProtectedBlockLength)
{
    if (ProtectedBlockLength<25)
          Remainder3 = mod(PretectedBlockLeng, 3);
   else
          Remainder3 = mod(PretectedBlockLeng-1, 3);

    if (Remainder3==0)
          return 0;
    else
          return (3-Remainder3);
}
```

– where mod(x, y) is to compute the remainder of dividend x and divisor y.

$$mod(x,y) = x - floor(x/y)*y$$

– floor(x) returns the largest integer not more than x

codeword filling table

ProtectedBlockLength (65-bit blocks)	Shortened Last Codeword(mixed)			Parity(65-bit blocks)			ParityBlockLength (65-bit blocks)	ProtectedBlockLength after padding (65-bit blocks)
	#Long	#Medium	#Short	#Long	#Medium	#Short		
0<ProtectedBlockLength<=12	0	0	1	0	0	5	5	3*ceil(Protectedlength/3)
12<ProtectedBlockLength<=24	0	0	2	0	0	10	10	3*ceil(Protectedlength/3)
24<ProtectedBlockLength<=76	0	1	0	0	15	0	15	3*ceil((Protectedlength-1)/3)+1
76<ProtectedBlockLength<=88	0	1	1	0	15	5	20	3*ceil((Protectedlength-1)/3)+1
88<ProtectedBlockLength<=100	0	1	2	0	15	10	25	3*ceil((Protectedlength-1)/3)+1
100<ProtectedBlockLength<=220	1	0	0	29	0	0	29	3*ceil((Protectedlength-1)/3)+1

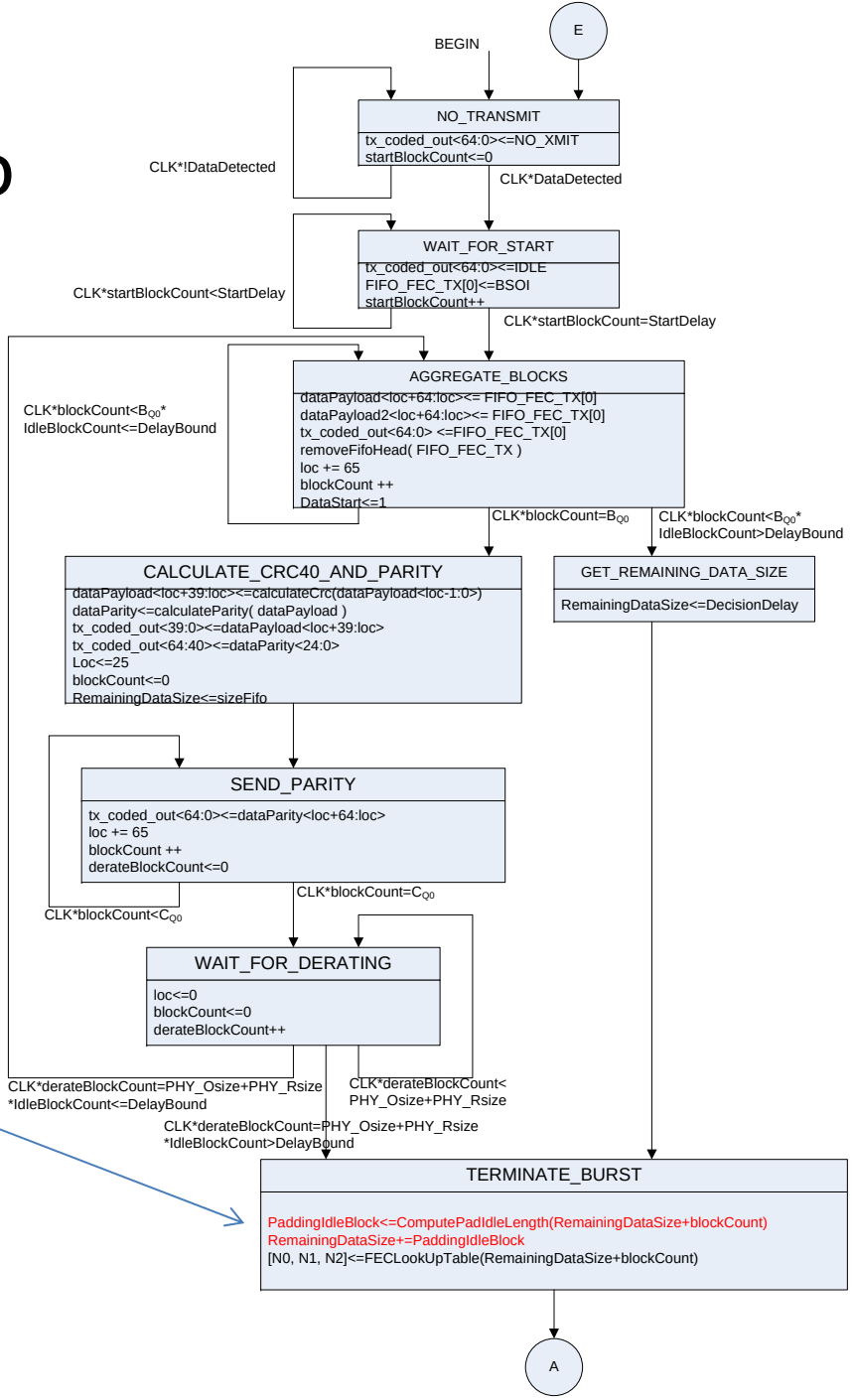
where ceil(x) returns the smallest integer not less than x

Solution to Burst End offset

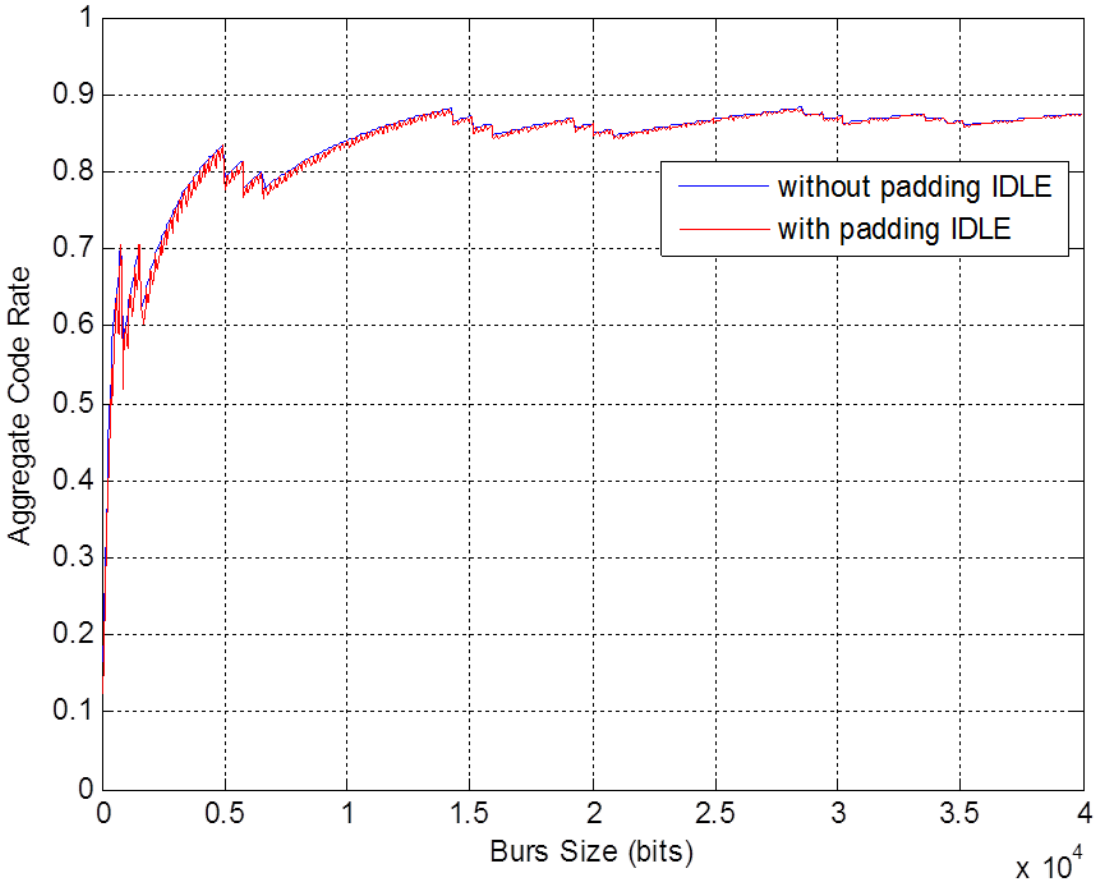
- This solution can be implemented through data detector output process.
- Simply “grab” Idle blocks from the DelayBound. No actual padding action is needed.
- The DelayBound is set to handle the worst case with certain margin. The solution only slightly increase the DelayBound (at most two 65-blocks without further optimization)
- The impact on coding efficiency is marginal considering the overhead of DelayBound

Data Detector Output Process with Solution to Burst End Offset

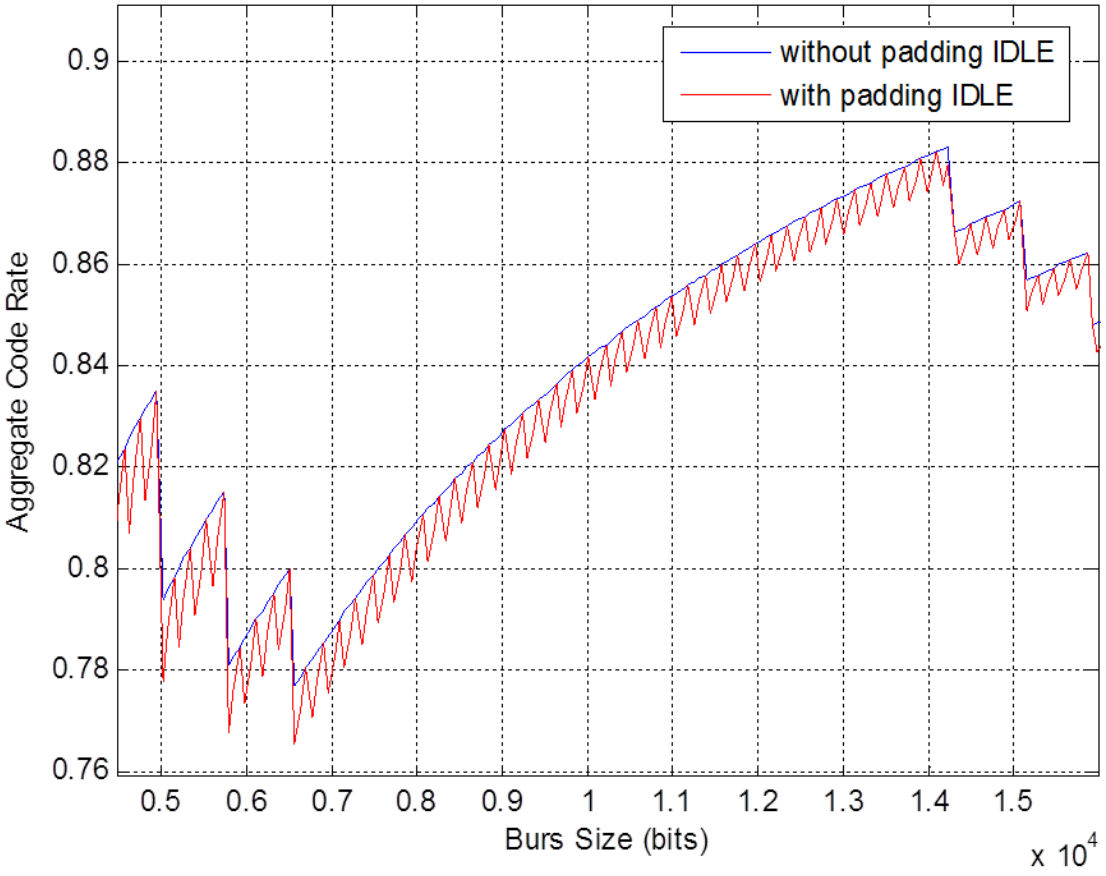
Handling burst end offset



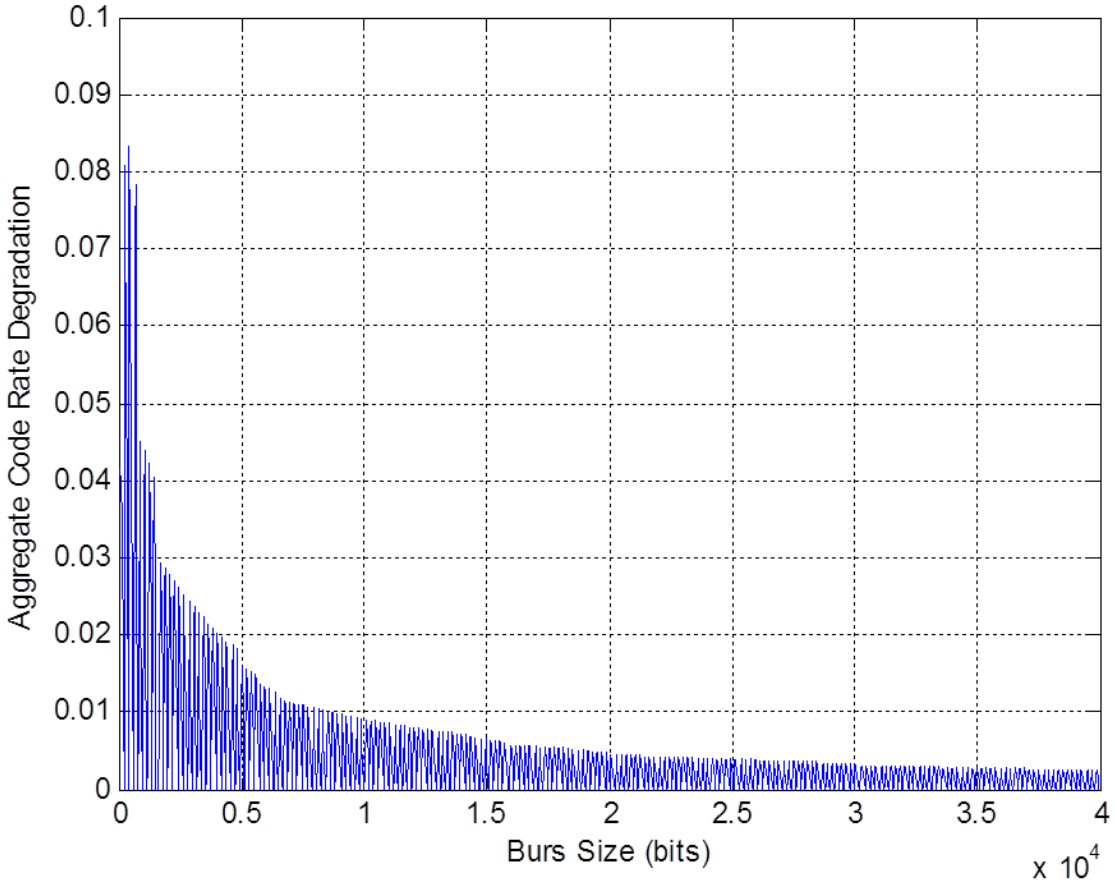
efficiency(Code Rate) vs. Burst Size(Information bits)



efficiency(Code Rate) vs. Burst Size(Information bits)



efficiency(Code Rate) degradation



efficiency(Code Rate) degradation