Programmable Pattern Generator For 10GBASE-R/W

Jonathan Thatcher

World Wide Packets

Motivation

- ***Motivation:*** **provide a simple to implement, programmable pattern generator.**
- ***Rationale:*** **it is not clear now, and may not be clear for a long time, what pattern provides the optimal characteristics for stressing a link (future graduate students beware ☺)**
  - **In fact, there may not be a single optimal pattern….**
  - **But, we have a recommendation that is a great starting point (see Ewen presentation)**

Requirements

- **Relatively short pattern**
  - **Able to be loaded into BERT memory**
  - **High repetition rate**
- **High degree of flexibility**
  - **Run length;**
  - **Disparity vs time;**
  - **Transition density;**
  - **Spectral content;**
  - **etc.**

Assumptions

- **MDIO registers are used to describe the algorithm. It is understood that the MDIO registers are optional and that an alternative method of control may be implemented.**
- **The bit stream indicated by the algorithm and its "seeds" is normative. There is no specific implementation prescribed or implied.**
- **While it should be clear that the concept used here could be applied to any PRBS, we use the 58 bit scrambler selected for the 10GBase PCS(s).**
- **We do not know how to embed this pattern within a SONET frame and ensure a relatively short, deterministic pattern "on the wire."**

Pattern -- General Description

- **Repetition of 4 sub-patterns of 2^{16} bits**
  - **Pattern0; Pattern1; Pattern2; Pattern3; Pattern0; Pattern2…**
- **Each sub-pattern is a segment out of the 2^{58} scrambler stream**
  - **Known starting state (last “64 bit” sequence of previous sub-pattern)**
  - **Modified to desired initial state by input of a previously calculated “psuedoseed” input to the scrambler**

Pattern options

- **May have quarter length pattern**
  - **Pattern0 = Pattern1 = Pattern2 = Pattern3**
  - **pSeed0 = pSeed1 = pSeed2 = pSeed3**
- **May have half length pattern**
  - **Pattern0 = Pattern2; Pattern1 = Pattern3**
  - **pSeed0 = pSeed2; pSeed1 = pSeed3**

Algorithm

1. **Load** -- Load Scrambler with bit Seed(0:63)
2. **Data** -- Shift in 1023 (2^{10}-1) sets of data(0:63)
3. **pSeed0** – Shift in 64 bit psuedoseed0(0:63)
4. **Data** -- Shift in 1023 sets of data(0:63)
5. **pSeed1** – Shift in 64 bit psuedoseed1(0:63)
6. **Data** -- Shift in 1023 sets of data(0:63)
7. **pSeed2** – Shift in 64 bit psuedoseed2(0:63)
8. **Data** -- Shift in 1023 sets of data(0:63)
9. **pSeed3** – Shift in 64 bit psuedoseed3(0:63)
   - **return to step 2.**

Transmit MDIO Registers/Bits

- **Transmit**
  - **Conformance Test Control (Normal; Test)**
  - **Test Data (0:63)**
  - **Seed (0:63)**
  - **PsuedoSeed0(0:63)**
  - **PsuedoSeed1(0:63)**
  - **PsuedoSeed2(0:63)**
  - **PsuedoSeed3(0:63)**

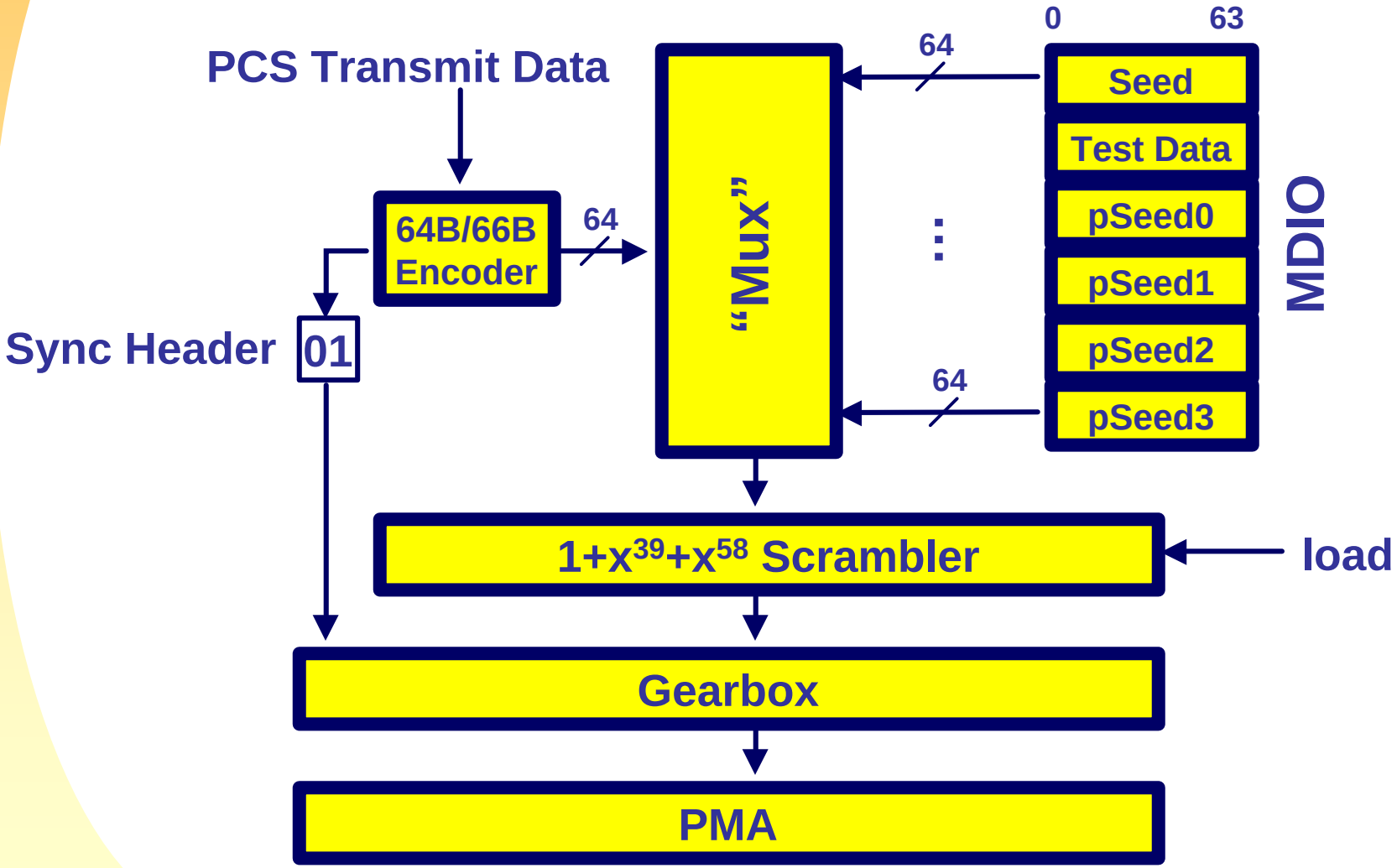
Receive MDIO Registers/Bits

- **Receive**
  - **Conformance Test Control (Normal; Test)**
  - **Test Data (0:63)**
  - **Seed (0:63)**
  - **PsuedoSeed0(0:63)**
  - **PsuedoSeed1(0:63)**
  - **PsuedoSeed2(0:63)**
  - **PsuedoSeed3(0:63)**
  - **Error Counter (0:15)**
  - **Error Counter Reset (clears Error Counter when written– auto-returns to 0)**

Layer Diagram

- **Note that the pattern is generated “after” the 64/66 encoder.**
  - **The only portion of the 64/66 encoder used is to insert the synchronization bits**

Pattern Generator Conceptual



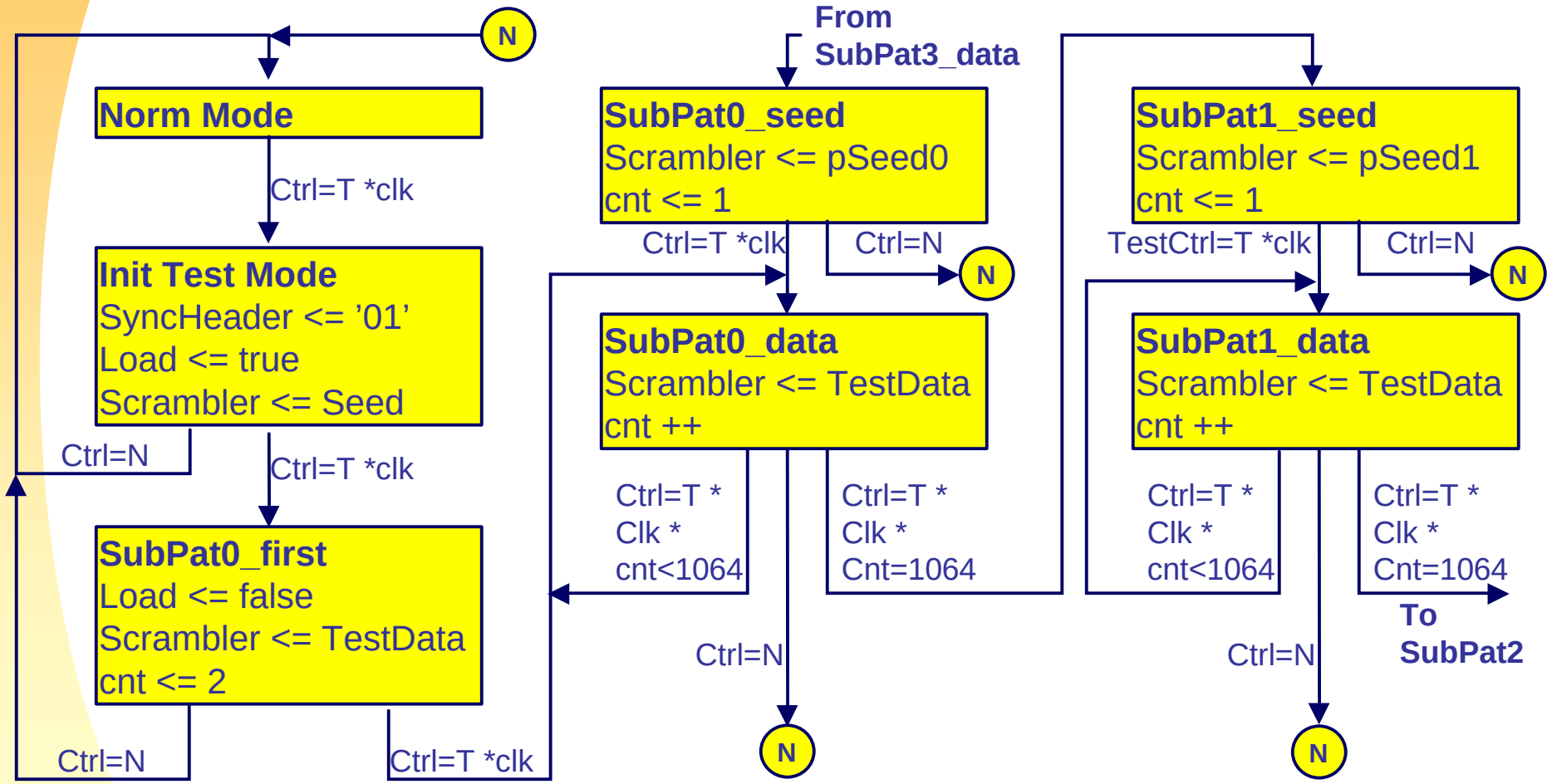
Sync Header

- **Assumed constant throughout compliance testing**
  - **If not, we need a specific algorithm that defines state such that spectrum is deterministic**
  - **Will be used by the Rx synchronization state machine to align on 66 bit (64 bit) boundaries**

Creating the Seed / Psuedoseeds

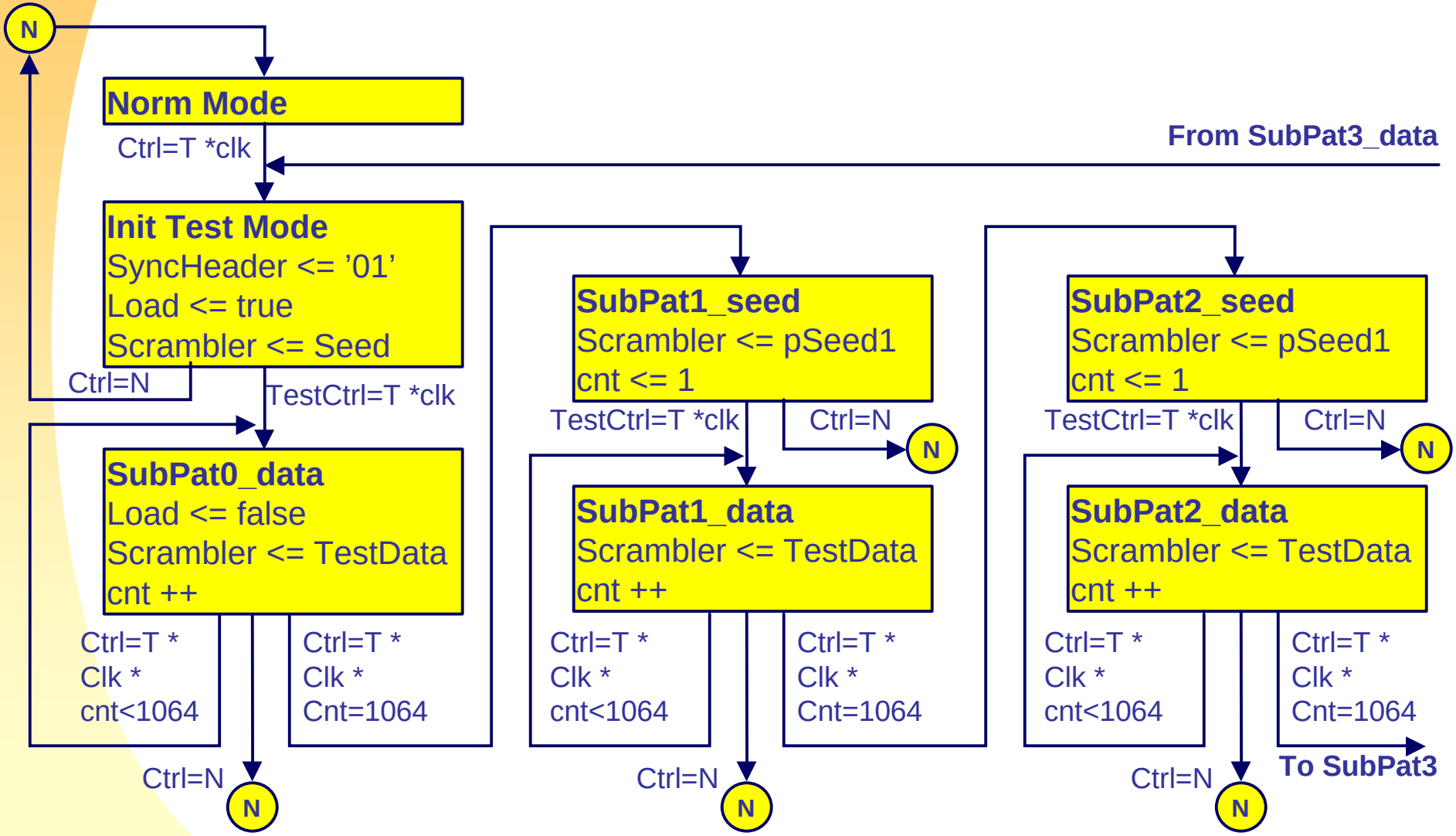
- **Seed:**
  - **64 bits**
  - **Equal to 58 bits loaded as seed**
  - **Plus 6 bits of predetermined prepend**
- **Psuedoseeds**
  - **64 bits….**
  - **Mathematically determined (deterministic) based on the result of the last state of the previous subpattern**

Pattern Generator State Diagram

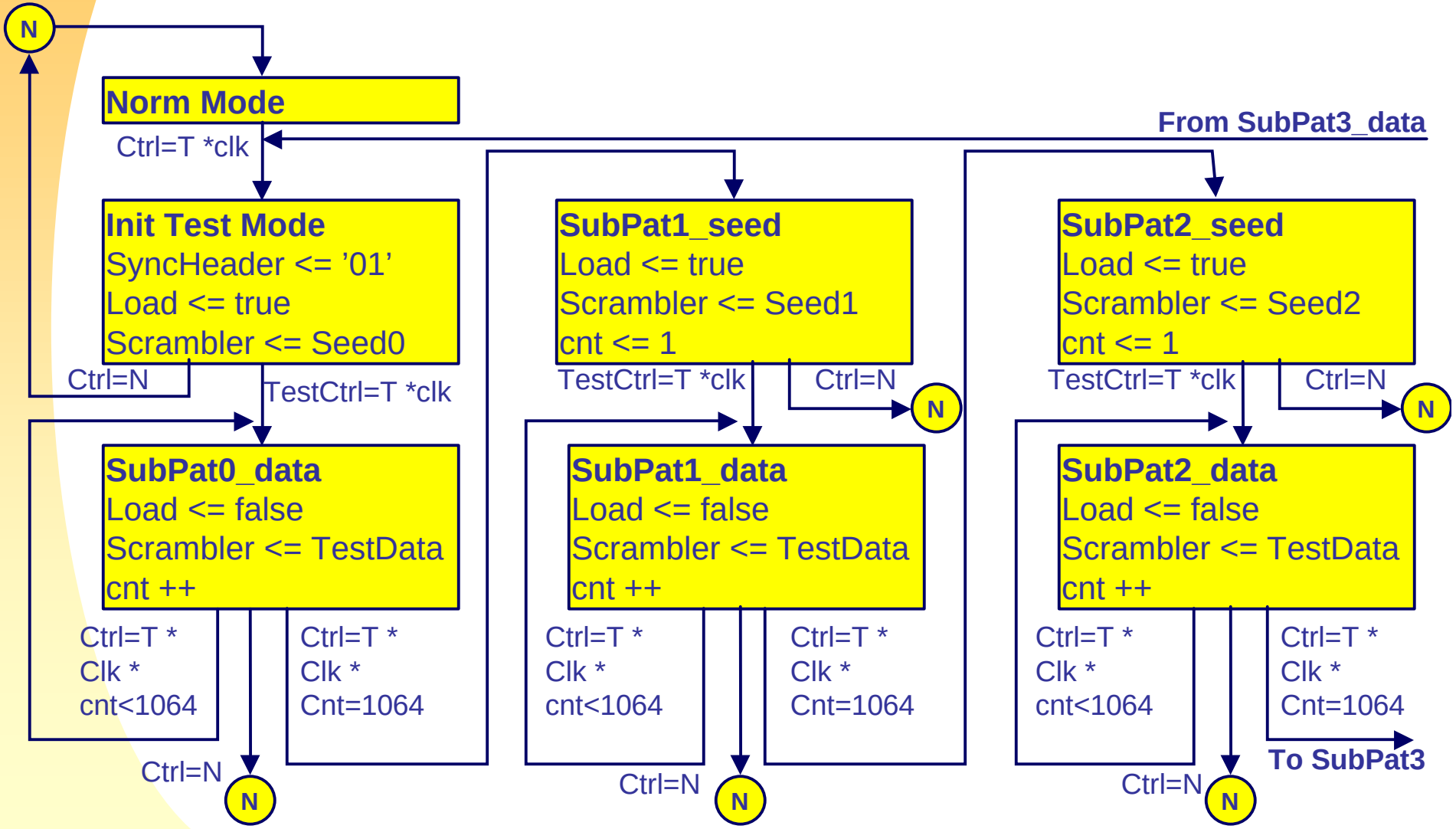


Not shown: SubPat2_seed; SubPat2_data; SubPat3_seed; SubPat3_data

Pattern Generator State Diagram 2



Pattern Generator State Diagram 3



Advantages of Pattern Generator 2

- **As compared to pattern generator 1…**
- **Guaranteed reset to known state at beginning of every pattern**
- **No PsuedoSeed0(0:63) register required**
- **Small simplification in logic**

Advantages of Pattern Generator 3

- **As compared to pattern generator 1…**
- **Guaranteed reset to known state at beginning of every subpattern**
- **No PsuedoSeed0(0:63) register required**
- **Seed0:3 used instead of pSeed0:3**
- **Small simplification in logic**

Receive Sync and Compare

- **Uses same algorithm as Tx for pattern**
- **Synchronization method needs to be determined by ad-hoc**
  - **Statistical**
  - **Simple state machine**
  - **Etc**
- **Check algorithm needs to be determined by ad-hoc**
  - **Bit by bit counter?**
  - **Word by word by word counter?**
  - **Reset during resync and under MDIO control?**

Motion

Move to create an ad-hoc to bring to the May meeting (with circulation 2 weeks before meeting) a complete draft of the (programmable) pattern generator described in (Ewen & Thatcher) / Thaler)

Moved:

Seconded:

Technical (75%):

For:

Against:

Abstain: