– SiLab Lecture Series –
Address Pads

Caleb Fangmeier

University of Nebraska - Lincoln

June 2, 2016

Outline

Addressing

Problem:

Many devices must communicate on shared lines.

Addressing

Problem:
Many devices must communicate on shared lines.

Why?

Addressing

Problem:
Many devices must communicate on shared lines.

Why?

- Limited physical space

Addressing

Problem:
Many devices must communicate on shared lines.

Why?

- Limited physical space
- Cost

Addressing

Problem:
Many devices must communicate on shared lines.

Why?

- Limited physical space
- Cost
- Simplicity of design

Addressing

Solution:
Addressing!

Addressing

Solution:
Addressing!

- Devices take turns talking on shared line

Addressing

Solution:
Addressing!

- Devices take turns talking on shared line
- Typically controlled by a single "**Master**"

Addressing

Solution:
Addressing!

- Devices take turns talking on shared line
- Typically controlled by a single "**Master**"
- All other devices are "**Slaves**"

Addressing

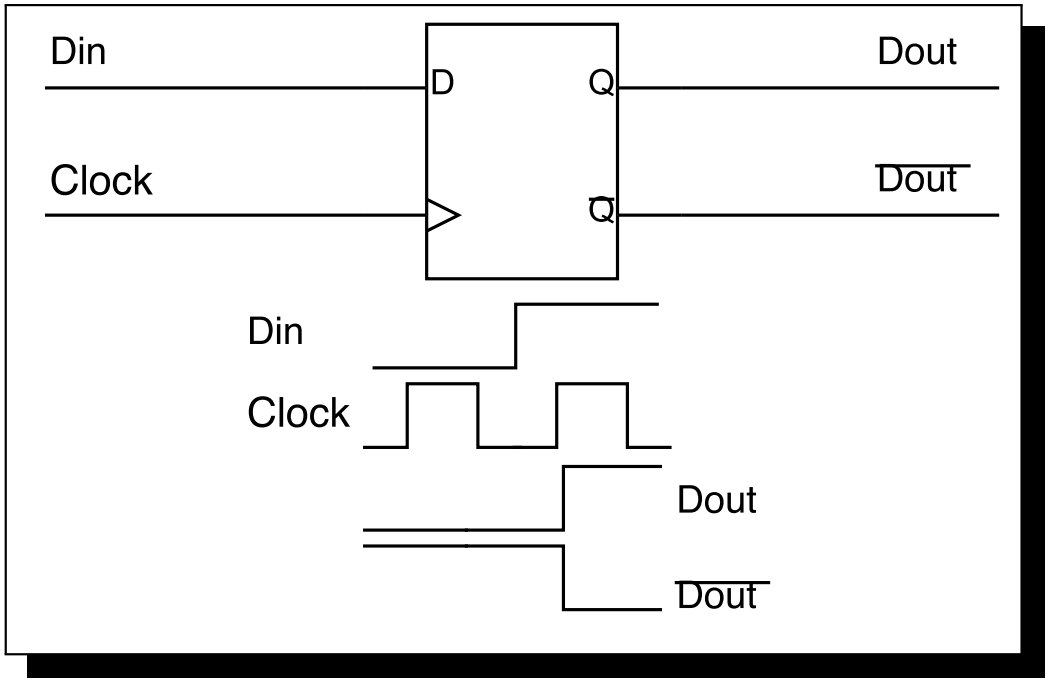
Solution:
Addressing!

- Devices take turns talking on shared line
- Typically controlled by a single "**Master**"
- All other devices are "**Slaves**"
- All **Slave** devices have an **address**

Outline

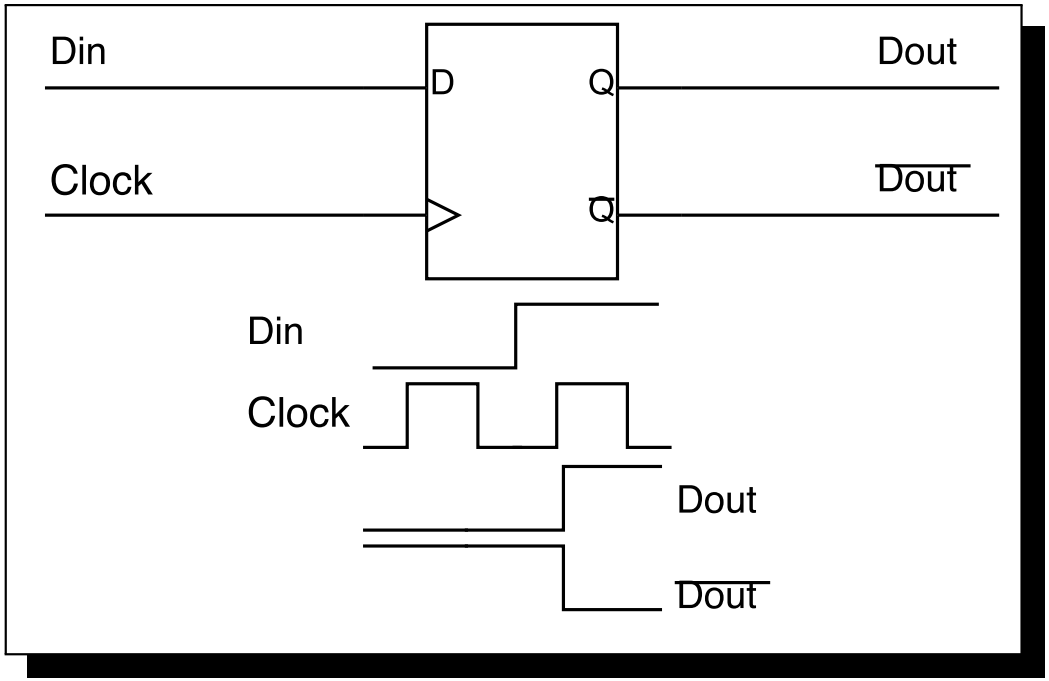
The "D" Flip-Flop

- Stores a single bit
- Has a **Data** input and a **Clock** input
- Data updates when clock transitions from low to high



The "D" Flip-Flop

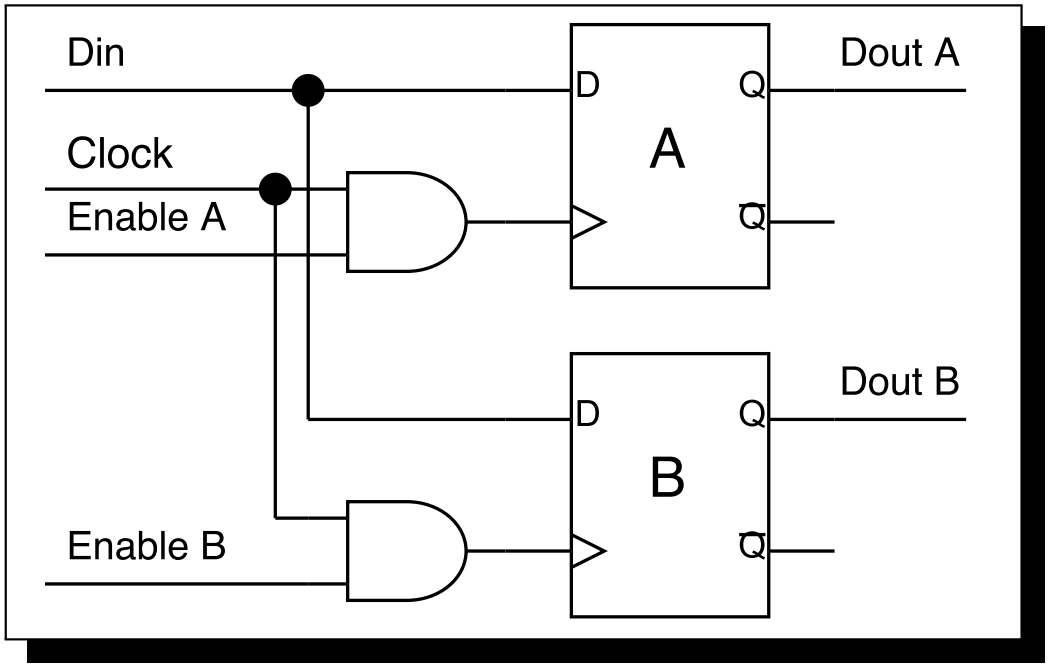
- Stores a single bit
- Has a **Data** input and a **Clock** input
- Data updates when clock transitions from low to high



But what if we want to control *multiple* devices with a shared data and clock line?

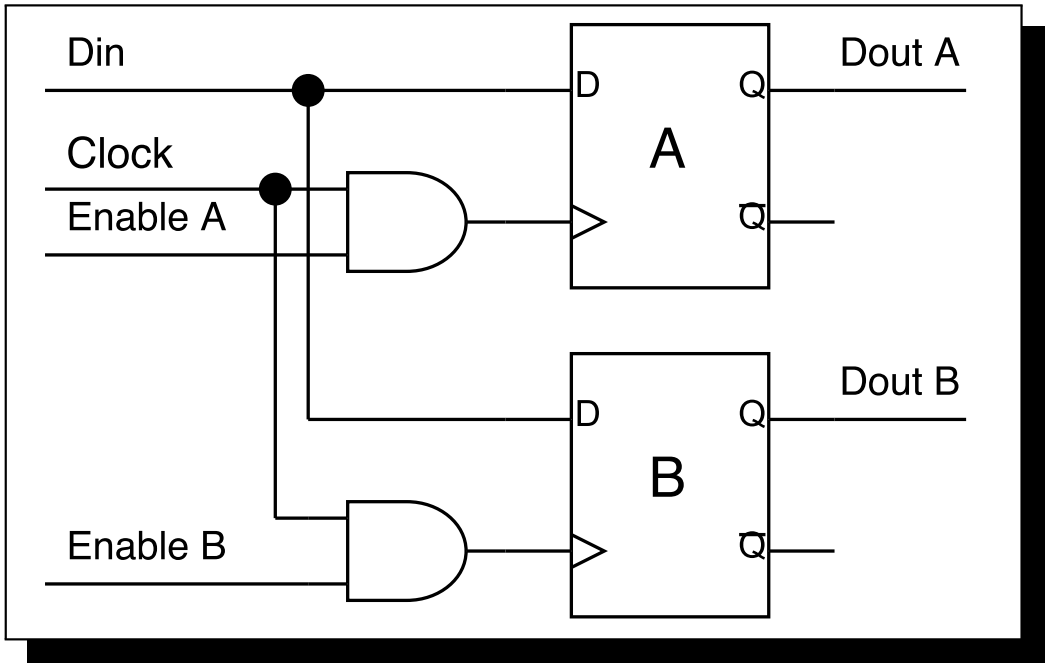
Shared Lines

- Add an **Enable** line for each device
- Requires $n + 2$ lines
- Data updates only when the **Enable** line is high



Shared Lines

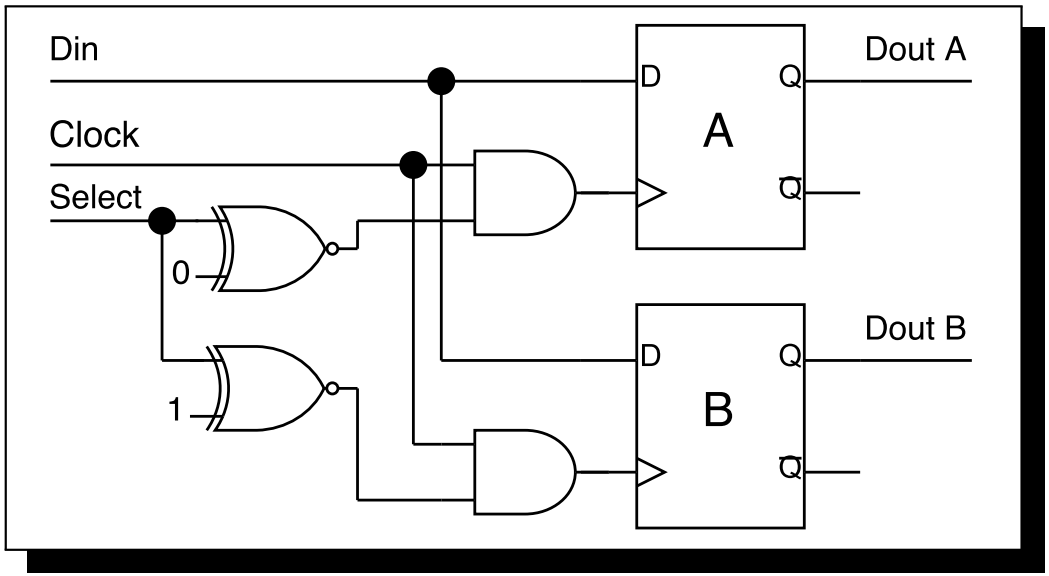
- Add an **Enable** line for each device
- Requires $n + 2$ lines
- Data updates only when the **Enable** line is high



Cool, but is this *optimal?* Short answer: It depends.

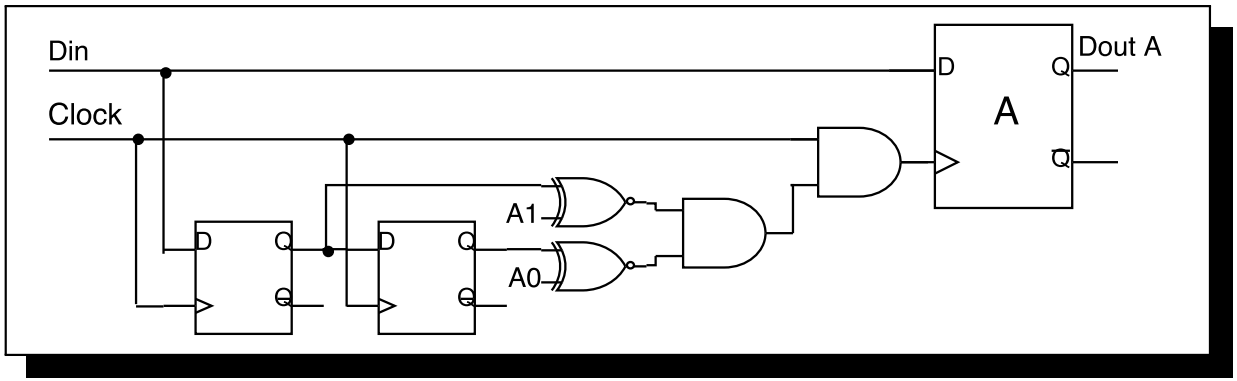
Address Lines

- Replace individual **Enable** lines with **Select**
- Requires $\lceil \log_2(n) \rceil + 2$ lines
- In this case **A** has address "0", while **B** has address "1"
- This scheme can be extened to any number of bits



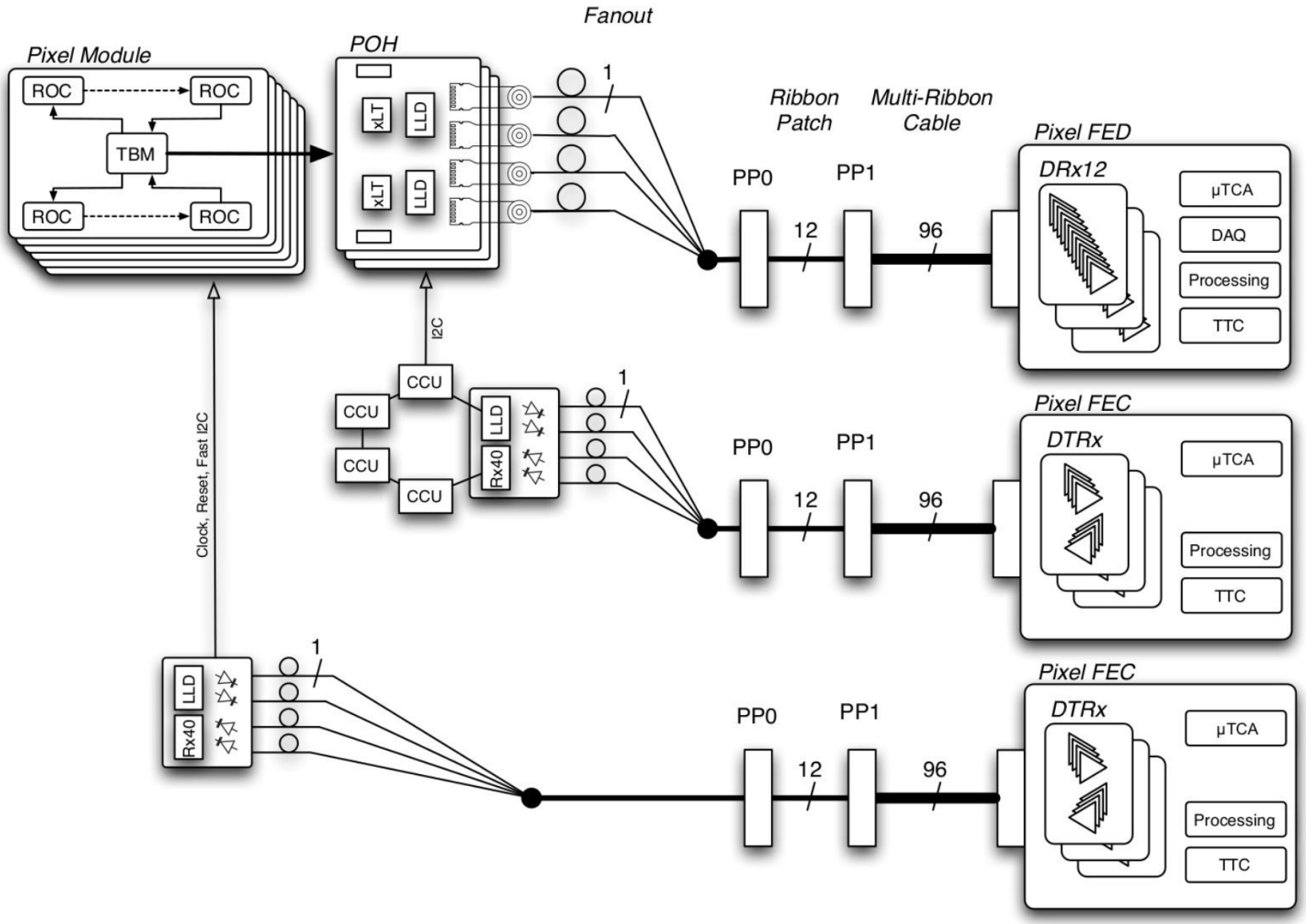
2-wire Protocol

- Address data is now pushed along data line
- Requires only 2 lines
- In this case **A** has a 2-bit address (A1,A0)
- This diagram is incomplete since it doesn't include end-of-write circuitry, but it demonstrates the idea.



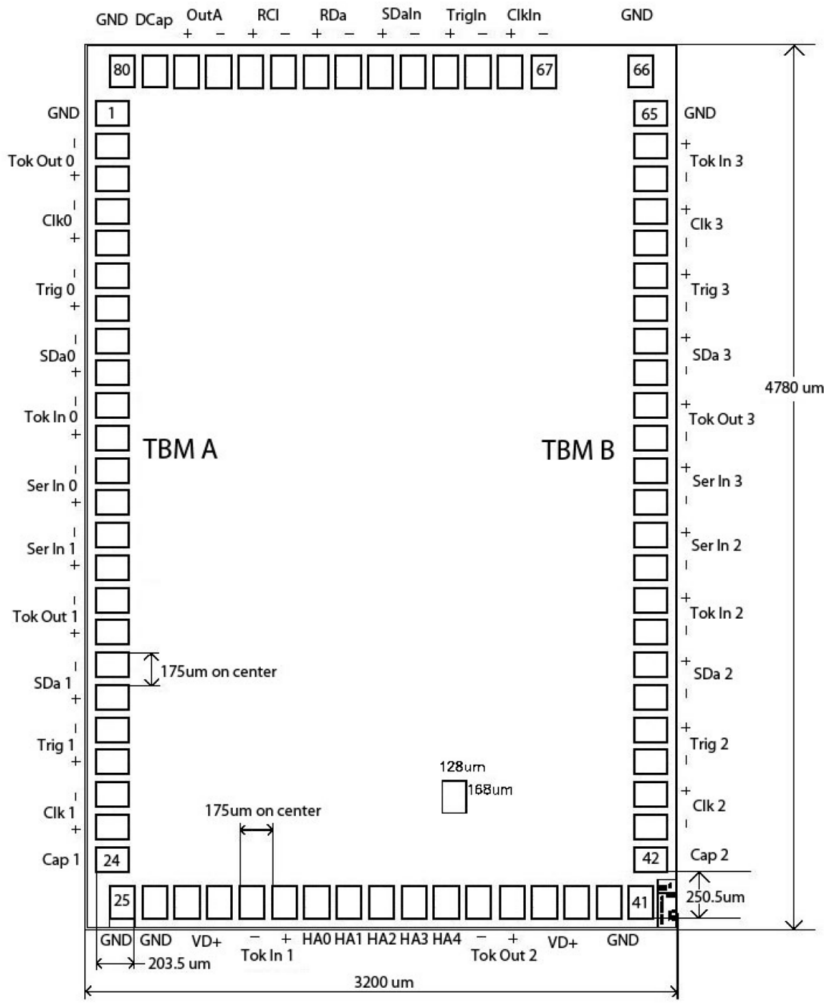
Outline

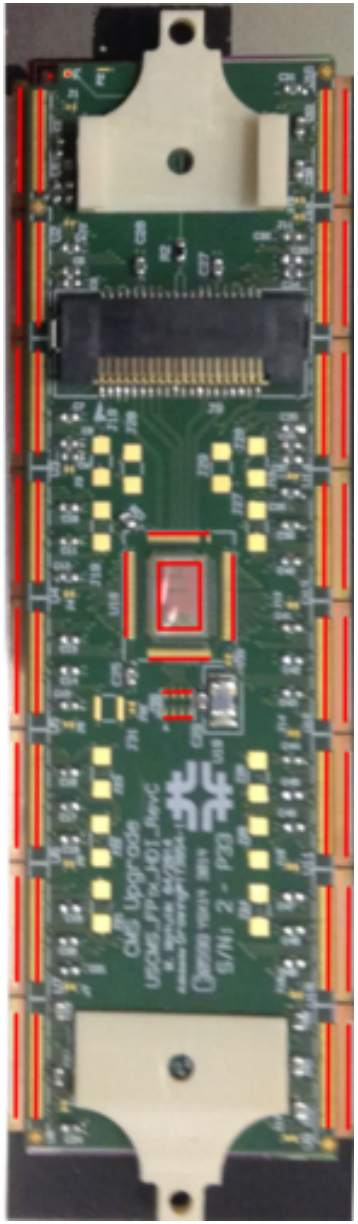
CMS Pixel DAQ

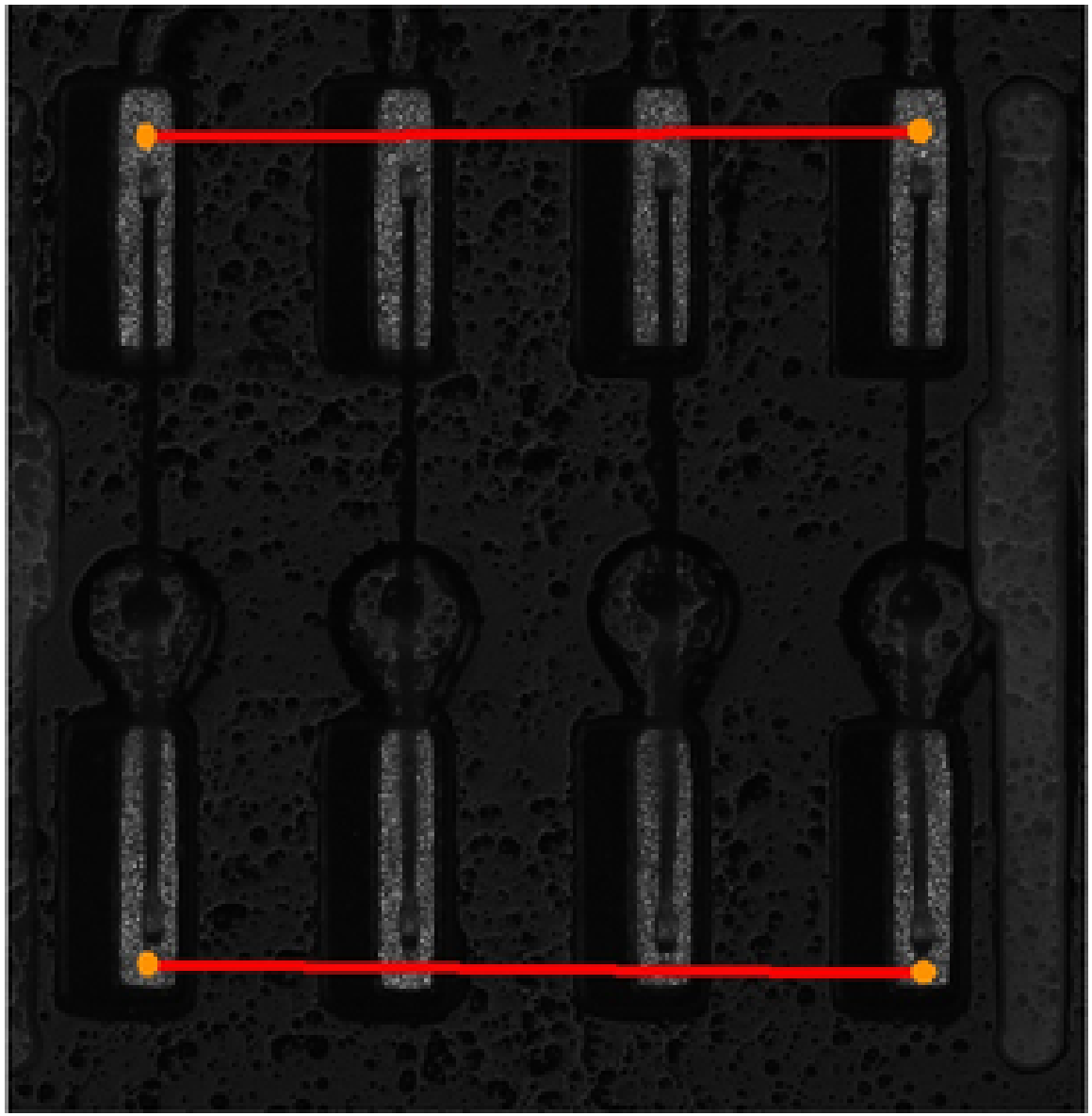


`HA*`	Hub Address Pads
	Internally pulled down
`ClkIn`	Serial Clock Input
`SDaIn`	Serial Data Input
`RCL`	Return Serial Clock
`RDa`	Return Serial Data

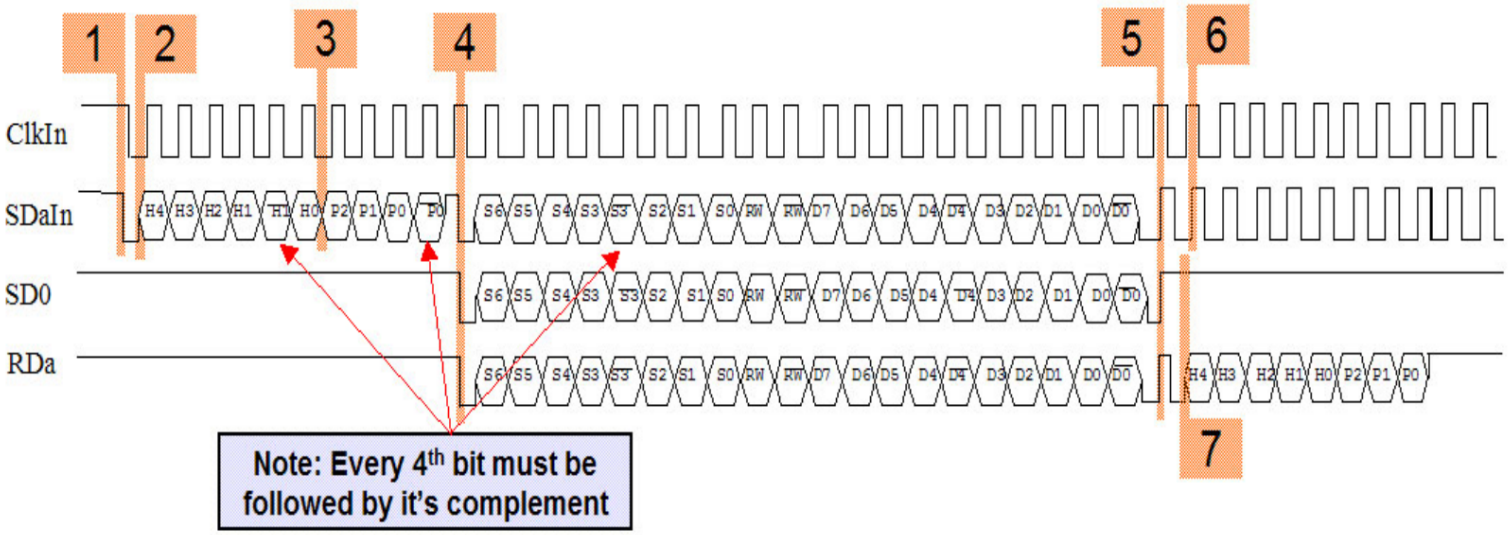
Note: The **data** readout does *not* happen on these lines. That is on `OutA`.







Serial Command Prototcol



1) Start Signal For Hub
2) Hub Address (H4, H3, H2, H1 H0)
3) Port Address (P2, P1, P0)
4) Start Signal for Readout Chip or Token Bit Manager. Note: Complimentary Bits Must Continue to be Sent During Command to ROC/TBM
5) Stop Signal to End Command to Hub
6) When inactive, Hub Should Continuously Receive Stops
7) Hub, and Port Address are Returned on RDa Line (Note: Complimentary Bits are not returned for Hub Address).

References

E. Bartz.

Token Bit Manager 08 Chip Documentation.
Rutgers University, Piscataway, NJ, 3 2013.

A. Dominguez.

CMS Technical Design Report for the Pixel Detector Upgrade.
Technical Report CERN-LHCC-2012-016. CMS-TDR-11, CERN, Geneva, Sep 2012.