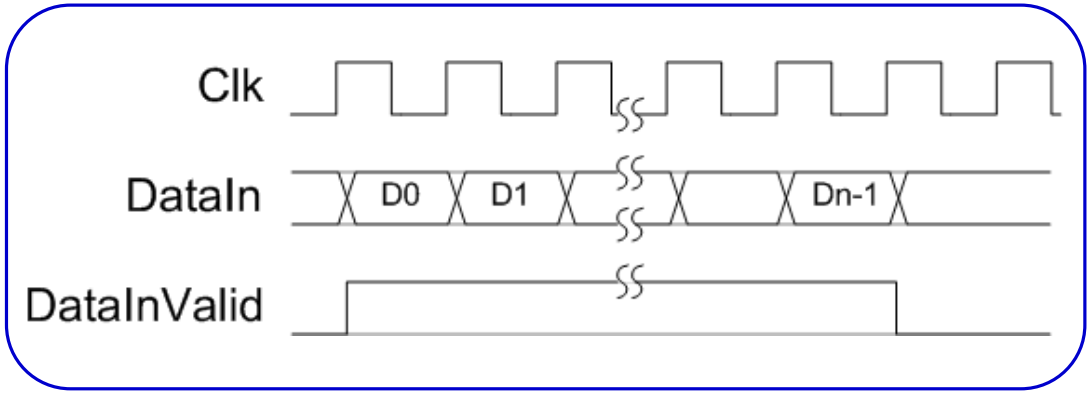
# DIGITAL DESIGN WITH FPGA CAMP

DAY 5 MEMORY

# OVERVIEW

# WHY MEMORY USED?



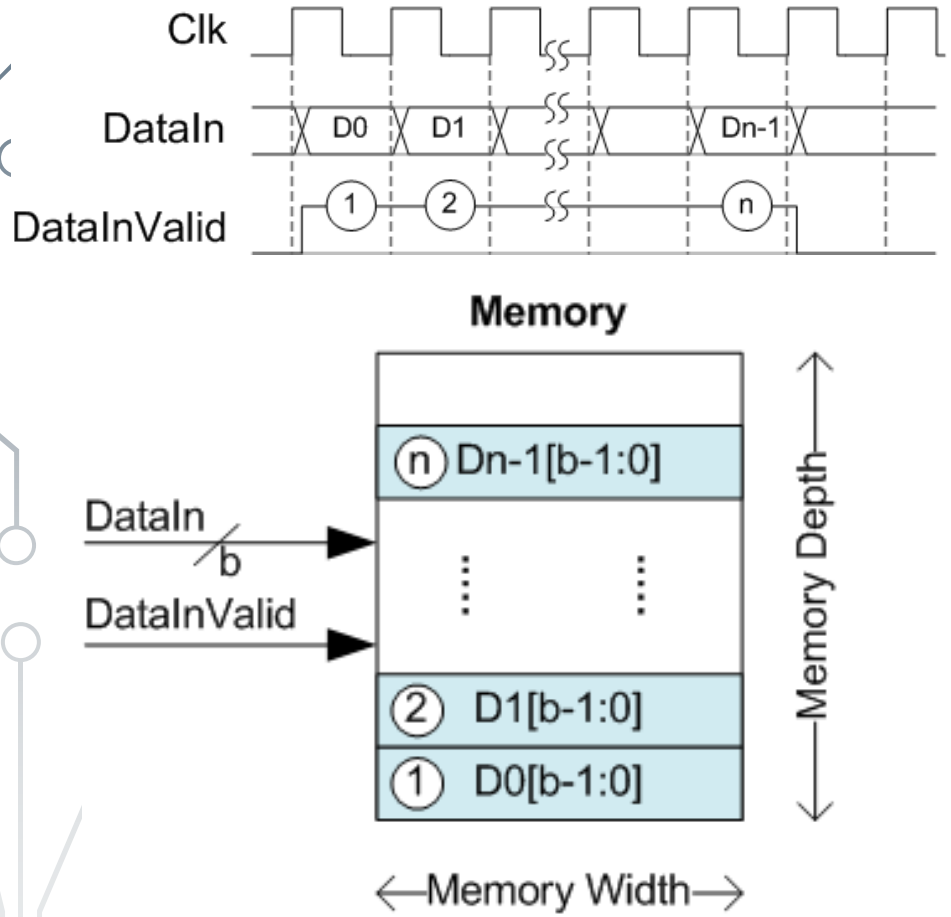

Q1: ข้อมูลขาเข้าถูกส่งมาอย่างต่อเนื่องจำนวนมาก หากจังหวะที่ข้อมูลถูกส่งเข้ามา วงจรยังไม่พร้อมรับข้อมูลใหม่ เพราะยังประมวลผลข้อมูลเก่าไม่เสร็จ ทำอย่างไร?

A1: นำข้อมูลใหม่ไปเก็บใส่หน่วยความจำไว้ก่อน แล้วค่อยนำมาประมวลผลทีหลัง

Q2: ใช้หน่วยความจำแบบไหนดี ขนาดเท่าไหร่ดี?

# MEMORY SIZE

Q: ใช้หน่วยความจำขนาดเท่าไหร่ดี?




A: การสร้างหน่วยความจำขึ้นมา ต้องมีข้อกำหนดอยู่ 2 อย่าง

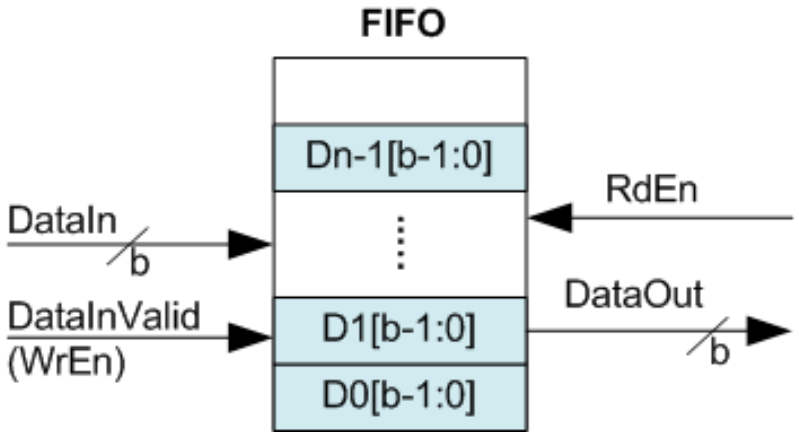
1) ความกว้างของข้อมูลที่จะเก็บ เช่น ขนาดของสัญญาณ DataIn ที่เข้ามามีขนาด b bit ดังนั้น width = b-bit เป็นอย่างน้อย เพื่อเก็บข้อมูลเข้ามาทีละตัว ในทุก ๆ clock cycle

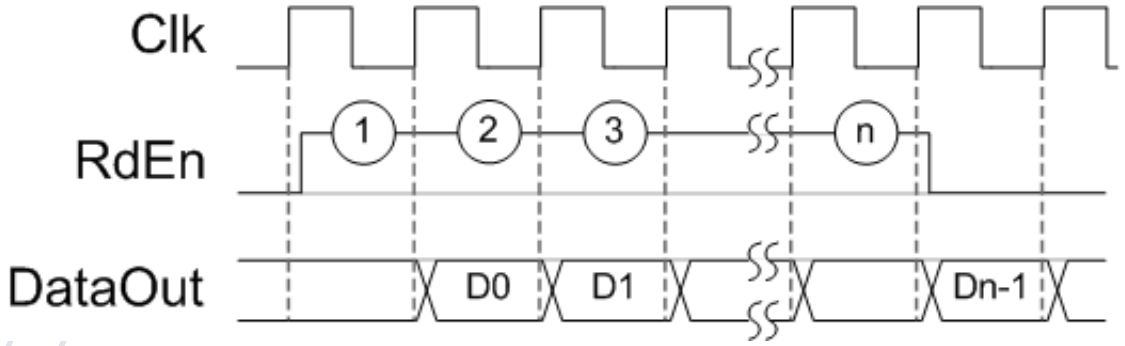
2) ความลึกของหน่วยความจำที่จะใช้ ขึ้นกับปริมาณข้อมูลที่จะต้องเก็บช่วงที่วงจรของเราไม่ว่าง เช่น ต้องรับข้อมูล n ตัว Depth = n เป็นอย่างน้อย

ข้อควรรู้!!! : ขนาดความลึกของหน่วยความจำ

มักจะกำหนดเป็นขนาด 2^N เช่น 32, 64, …, 512, 1024, …

# MEMORY TYPE (FIFO)



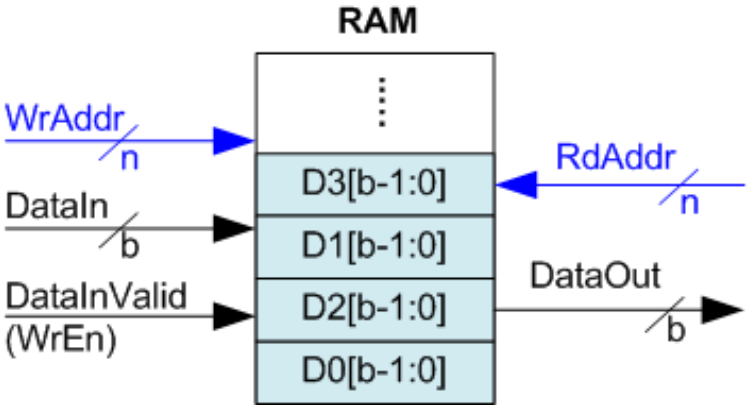

Q: ใช้หน่วยความจำแบบไหนดี?

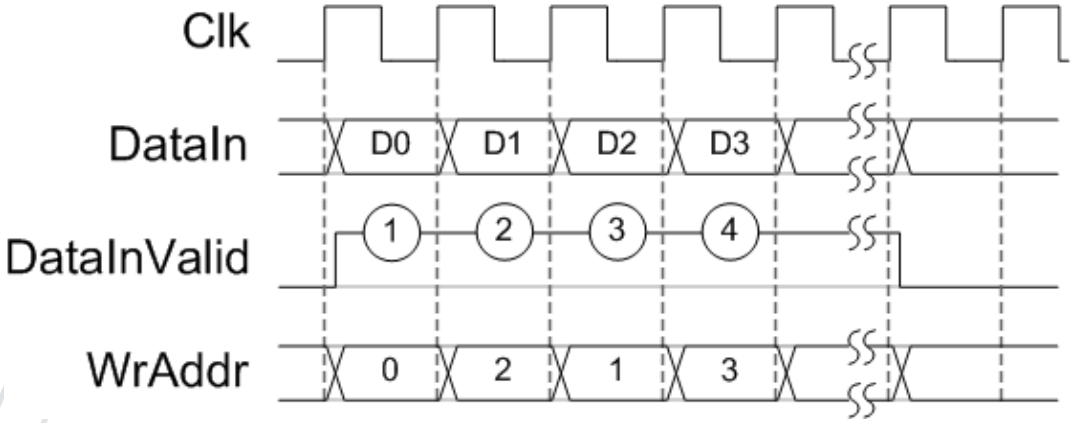
ลักษณะของ FIFO

- ลำดับข้อมูลด้านเข้า/เขียน (DataIn) และลำดับข้อมูลด้านออก/อ่าน (DataOut) เหมือนกัน (D0, D1, …, Dn-1)
- สัญญาณควบคุมจะน้อย คือ data และสัญญาณ enable เพื่อเขียนหรืออ่านเท่านั้น

FIFO เหมาะกับข้อมูลที่มีลำดับที่แน่นอนในการใช้เพื่อประมวลผล

# MEMORY TYPE (RAM)



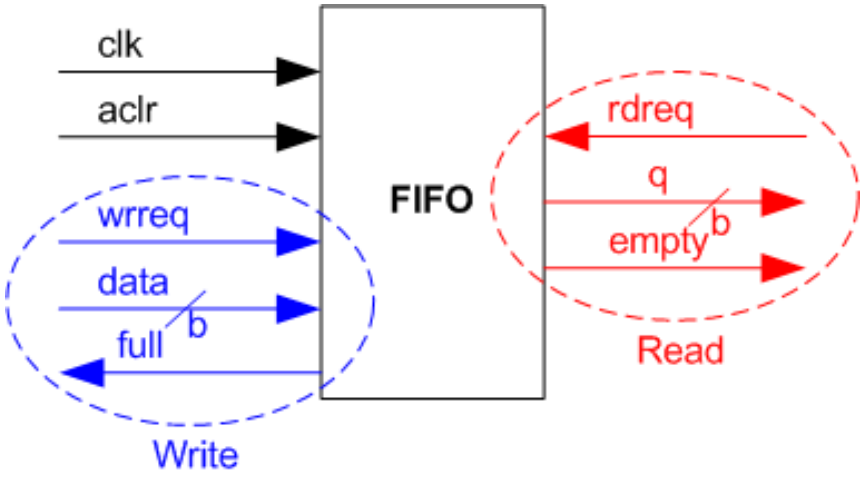




ลักษณะของ RAM

- ลำดับข้อมูลด้านเข้า/เขียน (DataIn) และลำดับข้อมูลด้านออก/อ่าน (DataOut) เป็นอิสระต่อกัน สามารถกำหนดได้
- การกำหนดลำดับของข้อมูลด้านเข้า/ออกนั้น ใช้สัญญาณ Addr เป็นตัวกำหนด
- ความกว้างของสัญญาณ Addr จะมีความสัมพันธ์กับ Memory Depth คือ เป็น n และ 2^n กล่าวคือ ถ้า Memory Depth = 32 สัญญาณ Addr จะมีขนาด 5 bit

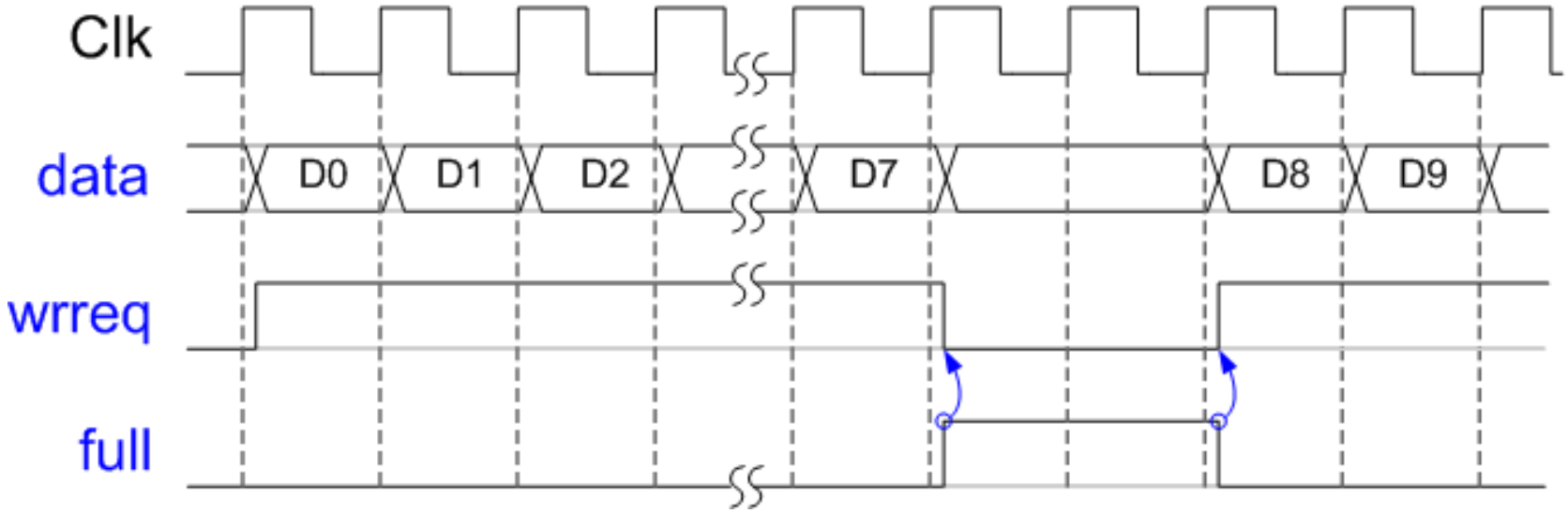
# FIFO

# FIFO INTERFACE



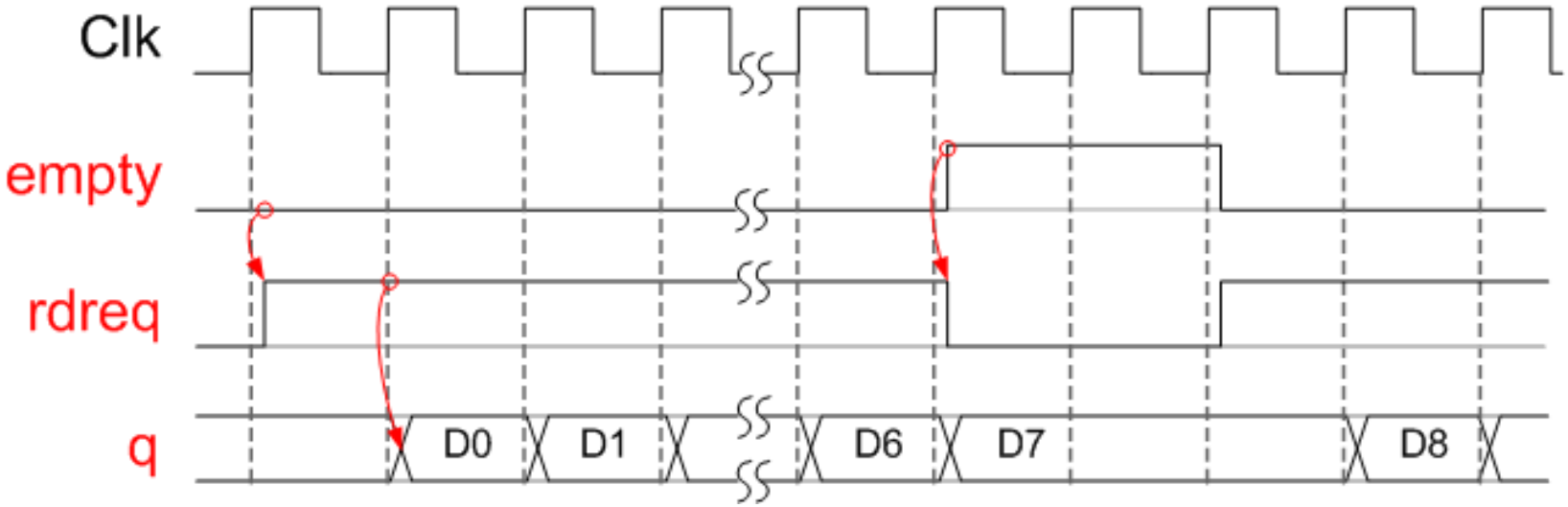

- clk: สัญญาณ input/output ทั้งหมดต้องทำงานภายใต้ clk
- aclr: asynchronous clear เมื่อมีค่าเป็น '1' ข้อมูลที่เก็บใน FIFO ทั้งหมดจะหายไป
- wrreq: มีค่าเป็น '1' เพื่อเขียนข้อมูลไปเก็บใน FIFO ทีละตัว
- data: ค่าข้อมูลที่จะเก็บ ต้องทำงานคู่กับ wrreq
- full: มีค่าเป็น '1' เพื่อบอกว่า FIFO เต็มแล้ว ไม่สามารถเก็บข้อมูลเพิ่มได้อีก
- rdreq: มีค่าเป็น '1' เพื่ออ่านข้อมูลออกจาก FIFO ทีละตัว
- q: ข้อมูลขาออกที่ถูกอ่านจาก FIFO จะทำงานคู่กับ rdreq
- Empty: มีค่าเป็น '0' เพื่อบอกว่า FIFO มีข้อมูลอยู่ภายใน พร้อมให้อ่าน

# FIFO TIMING DIAGRAM (WRITE)



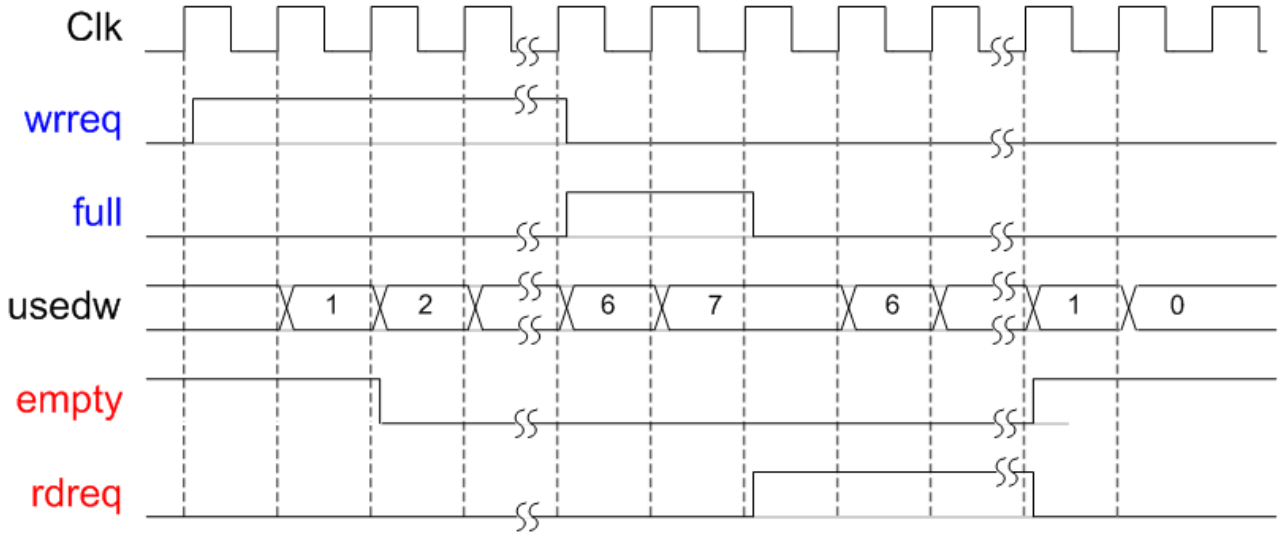

- เมื่อต้องการเขียนข้อมูลใส่ FIFO สร้าง wrreq=‘1’ พร้อมกับ data ที่ต้องการเขียน 1 ตัว/1 clock cycle
- เมื่อ FIFO มีข้อมูลเต็มความจุ full จะมีค่าเป็น ‘1’ สัญญาณ wrreq จะต้องเปลี่ยนเป็น ‘0’ ทันทีที่เจอ full=‘1’
- ดังนั้นสัญญาณ wrreq จะต้องถูกควบคุมด้วย full โดยไม่ผ่าน Flip-Flop เพราะต้องเปลี่ยนใน clock เดียวกัน

# FIFO TIMING DIAGRAM (READ)




- เมื่อต้องการอ่านข้อมูลจาก FIFO ต้องตรวจสอบ empty=‘0’ ก่อนว่ามี data เก็บอยู่ใน FIFO
- สร้าง rdreq=‘1’ เพื่ออ่าน data 1 ตัว/1 clock แต่ data (q) จะพร้อมให้อ่านใน clock ถัดไปหลังจาก rdreq=‘1’
- เมื่อ FIFO ไม่มีข้อมูลแล้ว empty จะมีค่าเป็น ‘1’ สัญญาณ rdreq จะต้องเปลี่ยนเป็น ‘0’ ทันทีที่เจอ empty=‘1’
- ดังนั้นสัญญาณ rdreq จะต้องถูกควบคุมด้วย empty โดยไม่ผ่าน Flip-Flop เพราะต้องเปลี่ยนใน clock เดียวกัน

# ADDITIONAL INFO#1 (USEDW)




- usedw เป็นสัญญาณที่บอกจำนวนข้อมูลที่มีอยู่ใน FIFO แต่ค่าจะ update ช้ากว่า empty หรือ full
- มักจะใช้สำหรับการเขียน/อ่านข้อมูลมากกว่า 1 ตัว เช่น วงจรที่ทำงานจะเขียน/อ่านข้อมูลทีละ 7 ตัว
  - วงจรเขียน จะดูว่า usedw นั้นเหลือพื้นที่มากกว่า 7 ตัวไหม แล้วเขียน 7 ตัวรวด
  - วงจรอ่าน จะดูว่า usedw นั้นมีค่ามากกว่าหรือเท่ากับ 7 หรือยัง แล้วจึงอ่าน 7 ตัวรวด
  - หลังจากเขียน/อ่านแล้ว 7 ตัวแล้ว วงจรอ่าน/เขียนจะหยุดทำงานชั่วคราว เพื่อรอให้ usedw นั้น update ค่าใหม่ให้เรียบร้อยก่อน ค่อยทำงานต่อ

# ADDITIONAL INFO#2 (ASYNCHRONOUS)

- FIFO สามารถเลือกให้วงจรเขียน/อ่าน ทำงานกันคนละ clock domain ได้ เรียก asynchronous FIFO
- FIFO จะมี clock input สำหรับวงจรเขียน (wrclk) และวงจรอ่าน (rdclk) แยกกัน
- สัญญาณ full และวงจรเขียนทั้งหมด จะทำงานอยู่ภายใต้ wrclk
- สัญญาณ empty และวงจรอ่านทั้งหมด จะทำงานอยู่ภายใต้ rdclk

# ADDITIONAL INFO#3 (DATA WIDTH/MODE)

- ความกว้างของ data ฝั่งเขียนและอ่าน สามารถ set ให้ต่างกันเป็น 2^n ได้
- การทำงานของ FIFO สามารถเลือก latency ฝั่งอ่านได้ (normal หรือ show-ahead)

อ่านบทความเพิ่มเติมที่

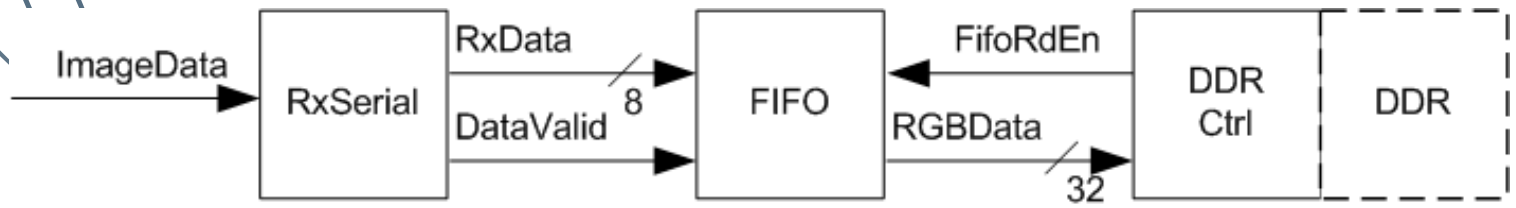
5.1 การทำงานของ FIFO บน FPGA

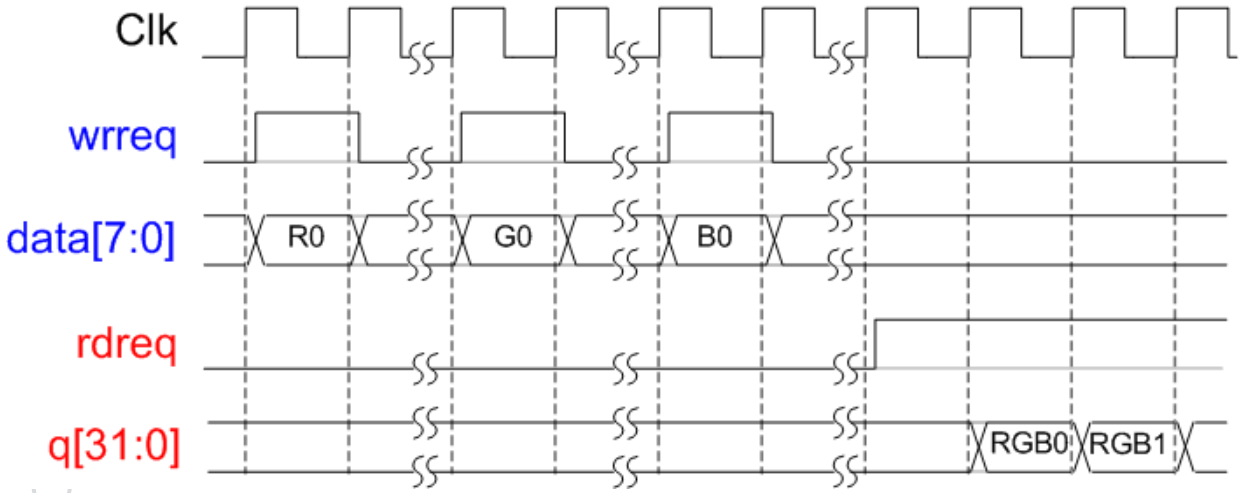
http://bit.ly/fpgafifo

5.2 ตัวอย่างการสร้าง FIFO โดยใช้ IP Catalog

https://forfpgadesign.wordpress.com/2017/11/19/5-2-fifo-example/

# FIFO APPLICATION



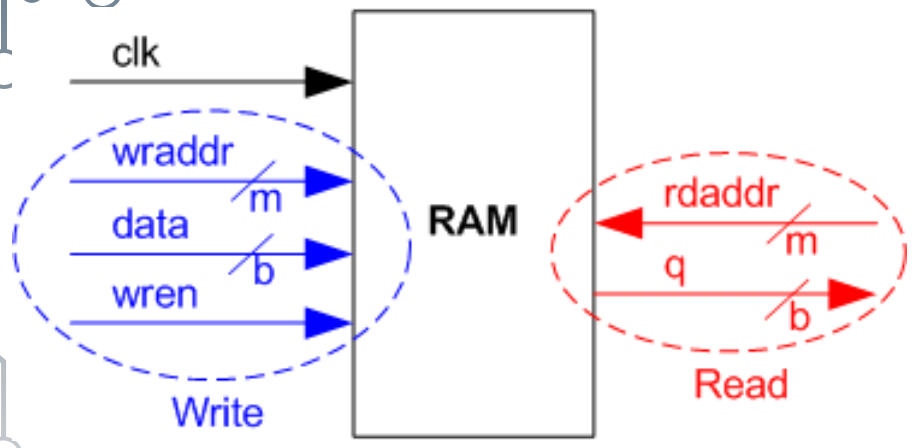




ข้อมูลภาพ RGB ถูกส่งมาผ่าน Serial ซึ่งจะรวบรวมข้อมูลมาเขียนทีละ byte ที่ FIFO โดยแต่ละ byte ก็จะห่างกันตามช่วงเวลาที่รับข้อมูลจาก Serial เสร็จ ดังนั้นข้อมูลจะถูกเขียนลง FIFO ทีละ 8-bit

เนื่องจาก DDR มีหลายวงจรร่วมกันใช้งานอยู่ และคุณสมบัติของ DDR คือต้องการรับ/ส่งข้อมูลครั้งละหลาย ๆ ตัวเลย ดังนั้นวงจรฝั่งอ่าน จะรอให้ FIFO มีข้อมูลมากเพียงพอแล้ว จึงอ่านทีเดียวหลาย ๆ ตัว โดยขนาดของ data ก็จะขยายเป็น 32-bit ซึ่งเป็นความกว้างของ data ที่ใช้รับ/ส่งกับ DDR

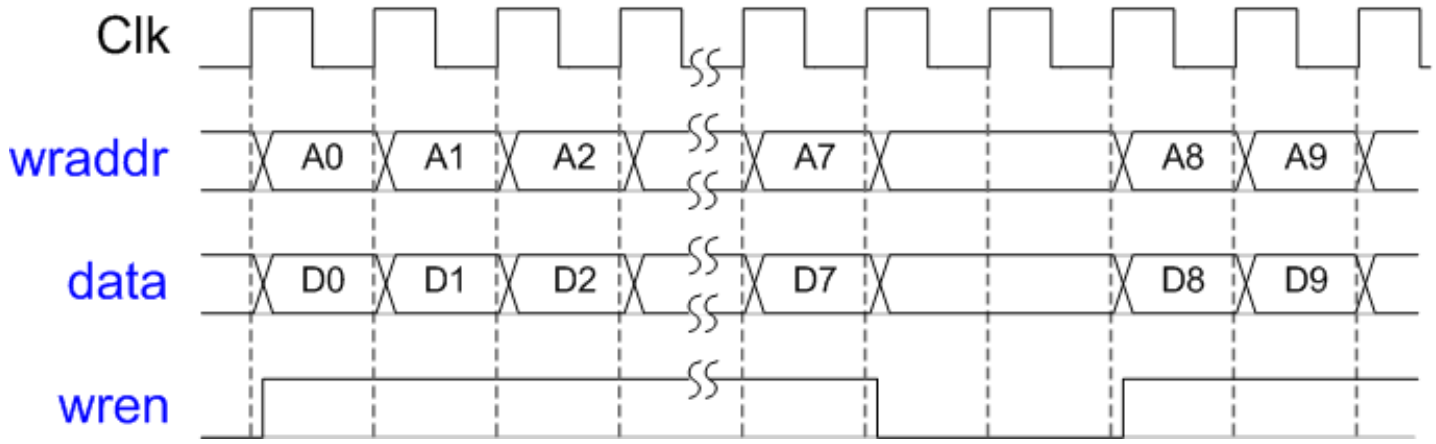
# RAM

# RAM INTERFACE



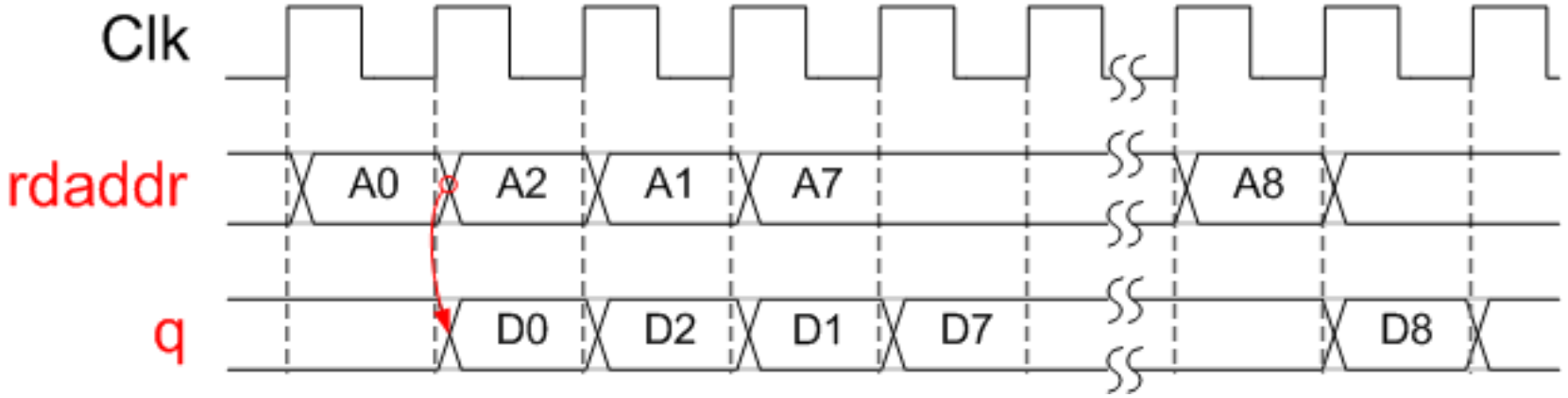

- clk: สัญญาณ input/output ทั้งหมดต้องทำงานภายใต้ clk
- wraddr: ระบุตำแหน่งที่จะเก็บ data ไปที่ RAM
- data: ค่าข้อมูลที่จะเก็บ ต้องทำงานคู่กับ wren และ wraddr
- wren: มีค่าเป็น ‘1’ เพื่อเขียนข้อมูลไปเก็บใน RAM ทีละตัว
- rdaddr: ระบุตำแหน่งที่จะอ่านข้อมูลจาก RAM
- q: ข้อมูลขาออกที่ถูกอ่านจาก RAM ซึ่งถูกควบคุมจากสัญญาณ rdaddr

# RAM TIMING DIAGRAM (WRITE)




- เมื่อต้องการเขียนข้อมูลใส่ RAM สร้าง wren='1' พร้อมกับ data ที่ต้องการเขียน 1 ตัว และกำหนด wraddr เพื่อระบุตำแหน่ง RAM ที่จะเก็บ ในแต่ละ 1 clock cycle
- ไม่มีสัญญาณบอกจำนวนข้อมูลที่เก็บไว้ใน RAM ดังนั้น user ต้องคำนวณเองว่าใส่ข้อมูลไปแล้วเท่าไหร่ ถูกอ่านออกมาแล้วเท่าไหร่ เพื่อจะได้ทราบจำนวนที่เหลืออยู่

# RAM TIMING DIAGRAM (READ)




- สร้างสัญญาณ rdaddr เพื่อระบุตำแหน่งของ RAM ที่ต้องการอ่านค่า
- ข้อมูลที่ตำแหน่งนั้น จะถูกส่งออกมาที่ q ใน clock ถัดไป
- ไม่มีสัญญาณระบุว่า RAM มีข้อมูลเก็บไว้ที่ตำแหน่งไหนบ้าง มีข้อมูลกี่ตัวแล้ว ดังนั้น user ต้องสร้างสัญญาณเพิ่มเติมเอง เพื่อคำนวณว่า RAM นี้มีข้อมูลพร้อมหรือยัง มีอยู่กี่ตัว และเก็บไว้ที่ไหนบ้าง
- เราจะใช้ RAM ทำงานในการที่ลำดับในการอ่านข้อมูลนั้นแตกต่างจากลำดับในการเขียนข้อมูล ซึ่งเป็นคุณสมบัติที่ FIFO ไม่รองรับ

# ADDITIONAL INFO#1 (EMBEDDED RAM)

- ภายใน FPGA จะมีหน่วยความจำสำเร็จรูปที่ฝังอยู่ ถ้าเป็น MAX10 จะเรียกว่า Embedded Memory ชนิด M9K ซึ่งแต่ละตัวจะมีขนาด 9K bit
- M9K 1 ตัวนั้นสามารถเลือกทำงานได้ 3 แบบคือ Single Port (เขียน/อ่านโดยใช้ address เดียวกัน), Simple dual port (เขียน/อ่านโดยใช้คนละ address ดังแสดงในตัวอย่างที่ผ่านมา), True dual port (มี port สำหรับให้วงจรสามารถเขียน/อ่าน ได้พร้อมกัน 2 วงจร)
- M9K สามารถโปรแกรมเลือกความกว้างของข้อมูลได้ เช่น x1, x2, x4, x8, x16 , x32 ทำให้ขนาดของ address จะปรับเปลี่ยนไปตามความกว้างของข้อมูล
- หาก RAM มีความลึกน้อย (ไม่เกิน 32) เราสามารถใช้ LE มาทำเป็น RAM แทน Embedded Memory ได้

# ADDITIONAL INFO#2 (ASYNCHRONOUS)

- RAM สามารถเลือกให้วงจรเขียน/อ่าน ทำงานกันคนละ clock domain (asynchronous FIFO)
- การทำงานของ 2-port RAM จะแบ่ง user เป็น 2 ส่วนคือ A และ B โดยทั่วไป
- ส่วน A จะเป็นวงจรเขียน RAM และส่วน B จะเป็นวงจรอ่าน RAM
- clock_a จะเป็น clock ควบคุมสัญญาณของการเขียน ทั้ง address_a (write), wren_a, data_a
- clock_b จะเป็น clock ควบคุมสัญญาณของการอ่าน ทั้ง address_b (read), q_b

# ADDITIONAL INFO#3 (DATA WIDTH)

- ความกว้างของ data ฝั่งเขียนและอ่าน สามารถ set ให้ต่างกันเป็น 2^n ได้
- การทำงานของ RAM สามารถปรับ latency ฝั่งอ่านได้โดยการใส่ FF เพิ่มเติม ทำให้ data ที่อ่านได้ จะออกมาช้ากว่า timing ปกติ 1 clock

อ่านบทความเพิ่มเติมที่

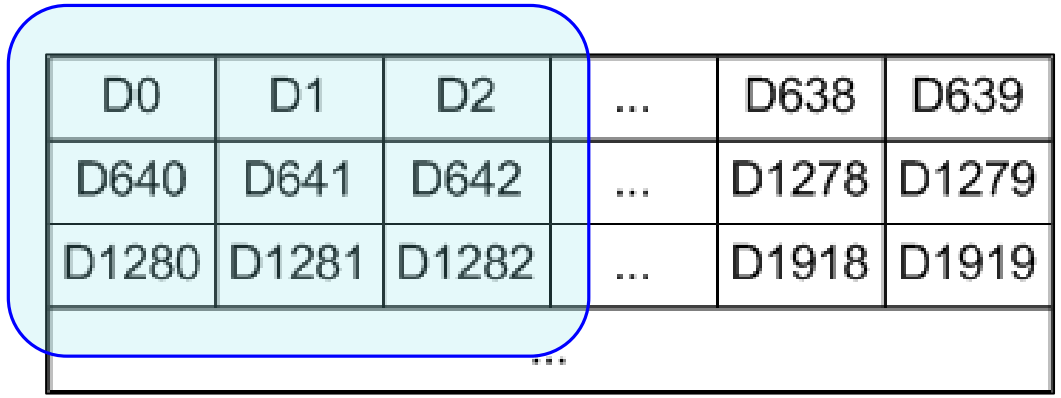
4.1 หน่วยความจำใน FPGA

http://bit.ly/fpgamem

4.2 ตัวอย่างการสร้าง RAM โดยใช้ IP Catalog

https://forfpgadesign.wordpress.com/2017/11/18/4-2-ram-usage-example1/

# RAM APPLICATION



| | | | | | |
|---|---|---|---|---|---|
| D0 | D1 | D2 | ... | D638 | D639 |
| D640 | D641 | D642 | ... | D1278 | D1279 |
| D1280 | D1281 | D1282 | ... | D1918 | D1919 |
| ... | | | | | |

การประมวลผลข้อมูลภาพในบาง application จะต้องนำ pixel รอบ ๆ มาประมวลผล เพื่อคำนวณค่าที่เหมาะสม ดังรูปคือ การนำข้อมูลทั้งหมด 9 ตัวมาคำนวณ ซึ่งจะเห็นว่า ข้อมูล 9 ตัวนั้นไม่ได้เป็นข้อมูล D0 – D8 ที่เป็นลำดับการเก็บข้อมูล แต่เป็น D0-D2, D640-D642, D1280-D1282 ดังนั้นลำดับการอ่านจะไม่ตรงกับลำดับการเขียน ทำให้ต้องใช้ RAM ในการช่วยเก็บข้อมูล แทนการใช้ FIFO

# Q & A

https://www.facebook.com/DigitalDesignThailand/

https://forfpgadesign.wordpress.com/