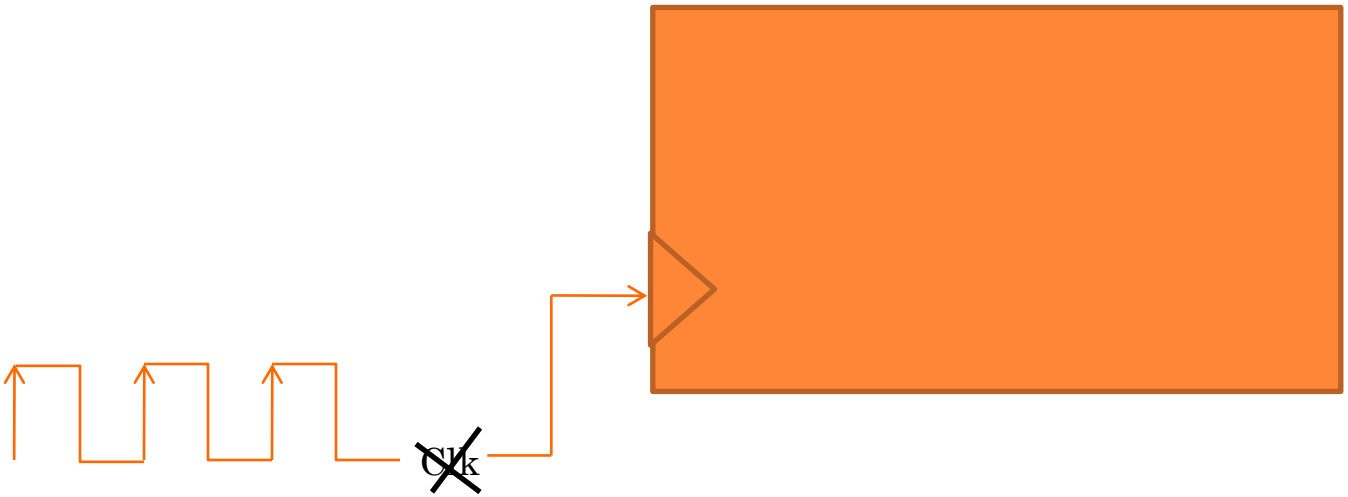
# Asynchronous and Self Timed Processors




**Presented By:**

**Zohaib Hassan**

# Agenda

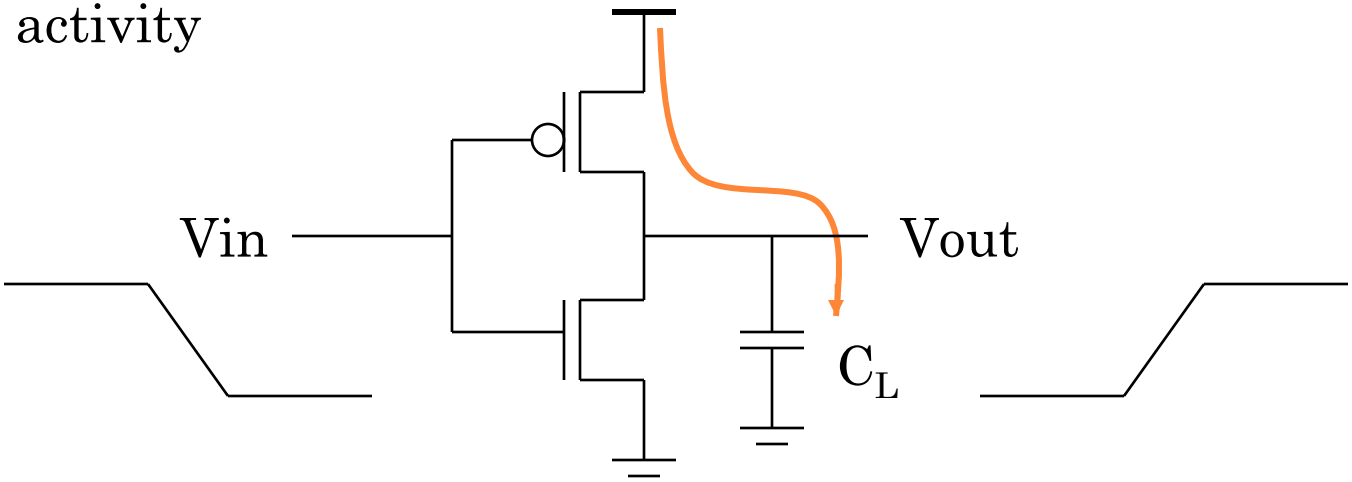
- Why Asynchronous design?
- Development of Asynchronous processors
- Asynchronous Design Styles
- Handshakes
- Features of Asynchronous design
- Asynchronous Reorder buffer
- Conclusion

# What Is wrong with the clocked processors ?

- Excessive Power  Dissipation Because All the State holding elements updated with clock
- Electromagnetic Interference
- All the Functions Operating at the same Rate
- Operating frequency a function of the slowest resource and critical path even if rarely used

# Power Consumption (Synchronous)

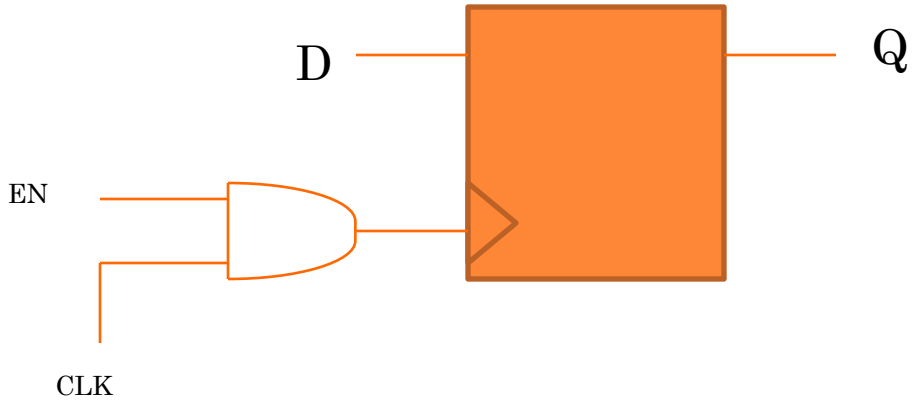
1. In CMOS circuits power consumption is a function of switching activity




$$P_{dyn} = C_{EFF} * V_{DD}^2 * f$$

As frequency increases, so does the dynamic power consumption of the CMOS transistors.

2. All the state holding elements are clocked regularly resulting in power dissipation, even if not desired
3. Up to 40% of the power budget expended on the Clock distribution
4. Clock gating can serve to reduce power consumption but it may lead towards undesirable skew if not managed properly.

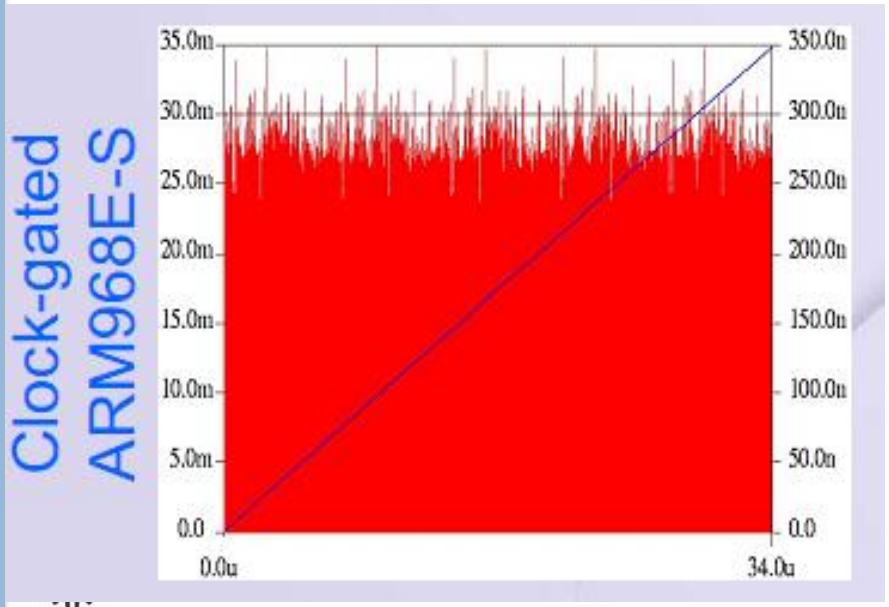


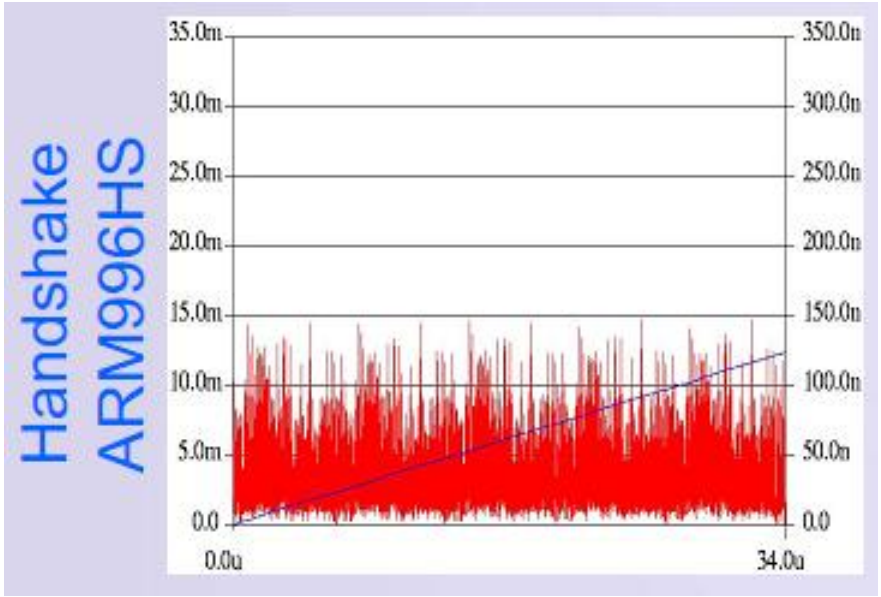
# POWER CONSUMPTION(ASYNCHRONOUS)

- Clockless processors are inherently “event driven”

  Process on demand otherwise STOP!

- Uses only the power required to do the job , Starts and shuts down instantaneously.



# Clock Distribution and Modularity

**Synchronous System:**

- Integration Levels increasing with time. (moore's law) Commercial IPs integrated together making up a complex SoC.
- Timing Closure i.e. getting all these devices to work at the desired clock becomes difficult.

NO CLOCK ! NO CLOCK DISTRIBUTION PROBLEMS

**Asynchronous Systems:**

• More convenient integration of different subsystems
•No problems of Timing closure.
•All the modules will operate as much as fast possible and no operation sets the timing for others ,exploiting maximum possible performance.
•Slow module will only slow down the system ,when used but no functional failure.

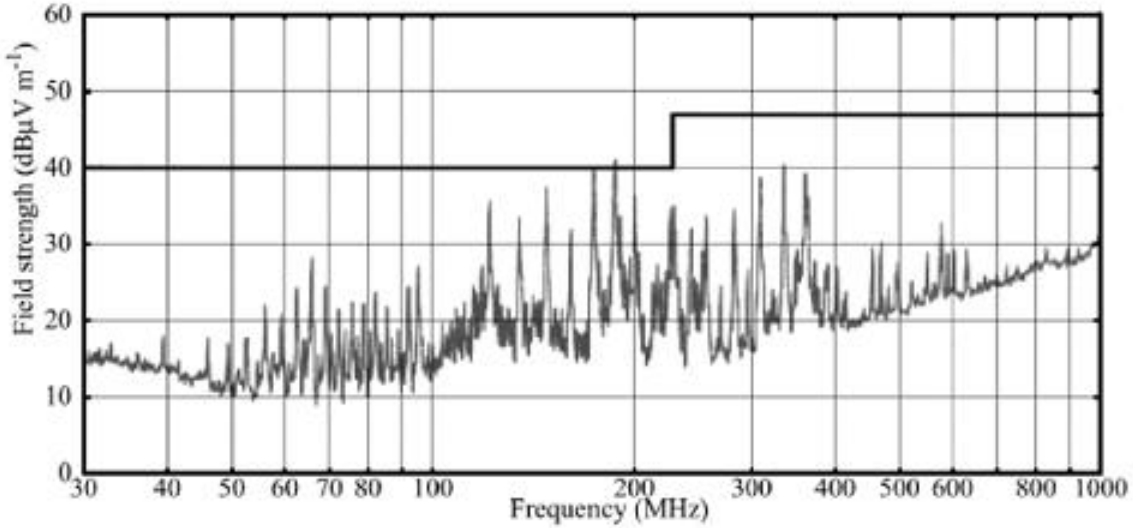
Currently GALS (globally asynchronous locally Synchronous) approaches But we focus on the extreme case ! i.e. no clock.
Clocked Modules communicating over asynchronous Bus.

# Electromagnetic Compatibility

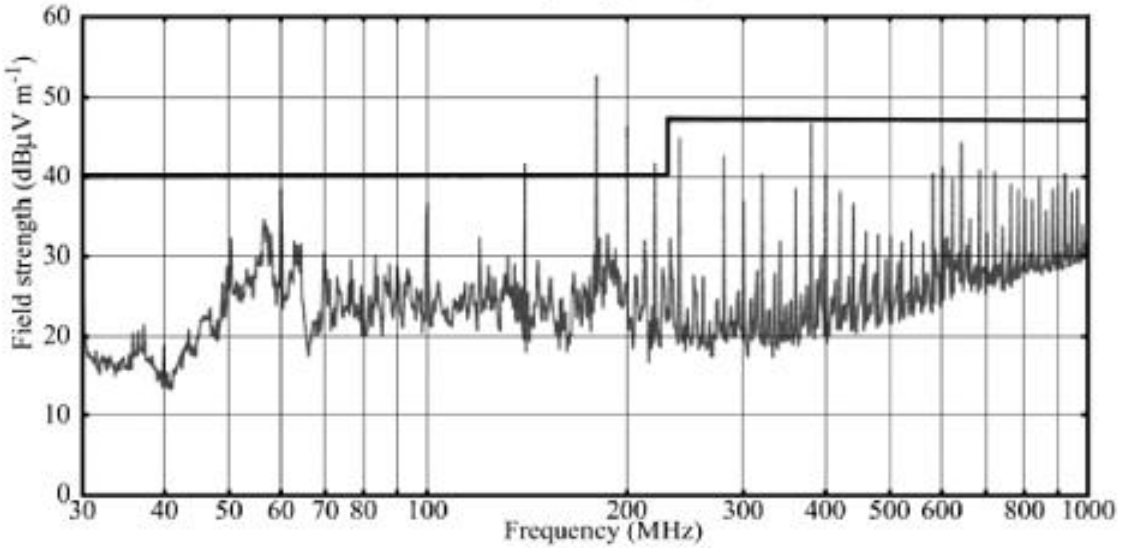
Synchronous Circuits:
• Peak current demands are correlated by the clock signal producing large and regular current spikes which are ideal for radiating radio frequency energy
➢ Electromagnetic compatibility needs to be addressed

Asynchronous Circuits:
• Less Electromagnetic emissions as switching activity is not the same , different modules operating at different speed and hence generating irregular current spikes and hence less electromagnetic emissions.
More suitable for applications involving Radio.

Amulet2




ARM6



# TRANSISTOR VARIATIONS

- More and more number of transistors are being packed into the single chip.
- Transistor strength is harder to predict as the variations in doping and gate length are large

Synchronous :
Frequency at which each chip will operate is harder to predict .Transistor Variations may lead to different propagation delays .

**Solution!**

Self Timed Logic:
As obvious from the name , self timed modules will only take time required To do the job and they set their own timings. No clock!

How to attract masses! No -GHz ?
How to measure performance?

# The DEVELOPMENT OF Asynchronous Processors

- **Early vacuum tube computers**
  - asynchronous in nature
  - Manual adjustments
- **Macromodule Project**
  - First systematic approach to develop some basic functional modules
  - System was robust, scalable and flexible in a way that it was tough to deliver the same with the synchronous technology of that time.
- **1st Asynchronous Processor**
  - "Caltech" in 1989 ,16 bit data path and almost all The functions supported expected of a microprocessor Excluding exceptions and interrupts.
- **Amulet1**
  - First MP that implemented commercial ISA developed by University of Manchester in 1993
  - ARM32 bit RISC reimplementation
  - Support for exceptions and interrupts
- **Amulet2**
  - Integrated in Amulet2e controller in 1996

Amulet3

- ARM9 class processor
- It was intended to be used in ISDN-DECT base because of its superior EM emissions compared with clocked equivalent.

❑ In parallel with Amulet1 development in 90s , a group at Tokyo Institute of Technology developed **TITAC** series of asynchronous microprocessors.

TITAC 1

8 bit MP with bespoke ISA

TITAC 2

Asynchronous re-implementation of MIPS

**•ALL these MPs were implemented using Full-Custom manual Layout Techniques.**

- In Late 90s ASIC and SoC were shifting from hardcore to synthesizable soft IP cores.

Lack of Synthesis Tools for asynchronous circuits . Future Of the clockless design tied with the future of the synthesis tools.

**Synthesis Tools development for Asynchoronous design**

**TANGRAM** asynchronous synthesis tool developed by PHILIPs in 1990s

• it was used to synthesize asynchronous 80c51 and it was quite Successful because it found excessive applications .

**BALSA** synthesis tool developed by Manchester group at the same time.

•It was used to develop fourth Amulet Processor SPA to exploit the contribution asynchronous logic could make to smart card security. ( Differential power Analysis and EMC Attack)

In 2006 ARM966 in the result of a collaboration between ARM and Handshake Solutions.

- ARM966 was developed with the Tangram(now Haste) Tools

  It could not meet clocked ARM9 in performance but it has respectable performance with excellent power efficiency.

**OBSERVATION:**

- From 89c51 to ARM966 successful asynchronous implementations but they were designed by keeping clocked processors in mind.

May be asynchrony offers an opportunity for a radical shift in ISA?

- Sun Labs CounterFlow pipeline architecture and FLEET communication oriented architecture are the attempts but not successful.

# Asynchronous Design Styles

- There are two main approaches practiced

1. Bundle-Data
2. Delay insensitive

**Bundle-Data**

- Many data wires "Bundle" with some asynchronous control signals
- Delay managed locally, All the circuit parts having their own timing models.
- Delay model can have different timing and delays for different operations performed by a module.

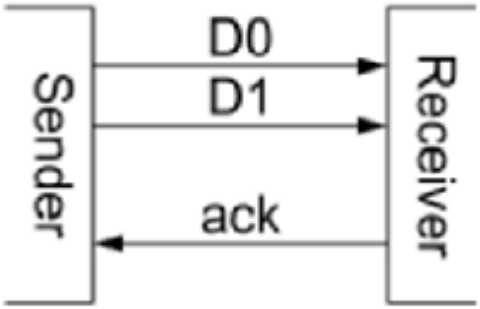
  ALU may perform addition with one delay and a logical operation With a different delay.

- How to produce a reliable delay model ?
- What if some arbitrary delay adds up due to some reason and computation takes more time than expected?

# DELAY INSENSITIVE

- Timing information embedded with the data.
- Circuit is guaranteed to function, no matter if some arbitrary delay adds up at any gate or wire.
- Purest realization of such circuits is extremely tough, but a closer approximation is **qDI** (quasi Delay insensitive)
- qDI assumes that isochronic fork is possible.

⇨ Dual rail encoding : each single bit encoded using two lines.




\+ immune from any variations in delay

\+ attractive for synthesis where wire delays may b unknow before layout

\- Roughly double silicone area then synchronous or bundle-data implementations.

# How to encode signals?

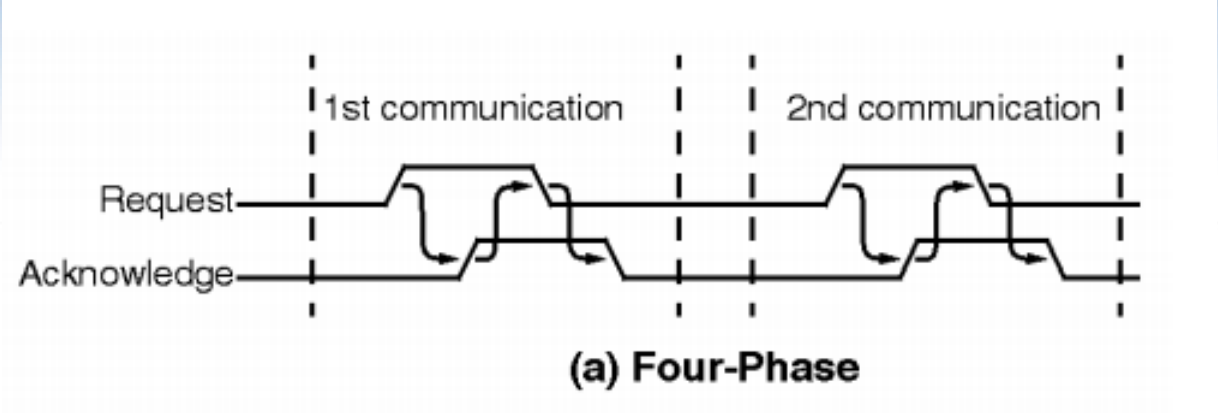
Level Sensitive

- Conventional way of assigning values
- Logic level 1 as high voltage and 0 as low.

Transition Sensitive

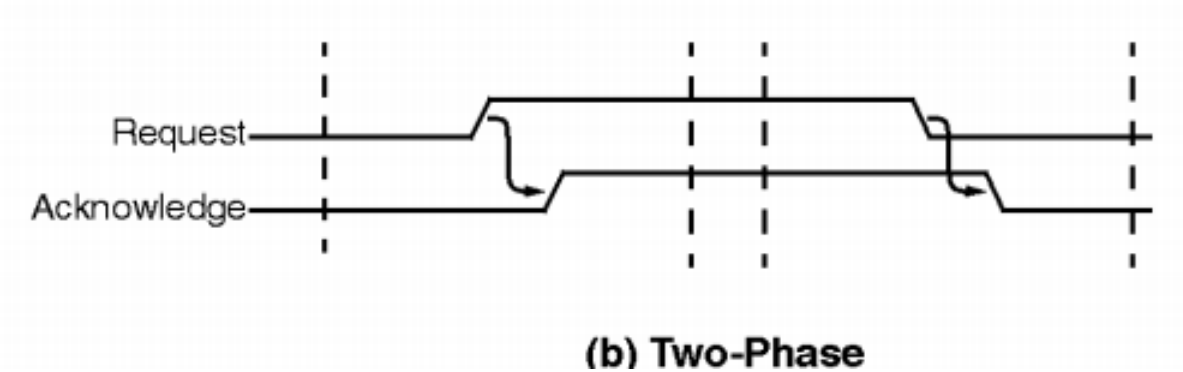
- A change in the signal either rising Or falling has the same meaning.
- Both transitions are called events.
- Avoids return to a neutral or low state
- Saves the time and energy cost of return Transition.

## Level sensitive handshake



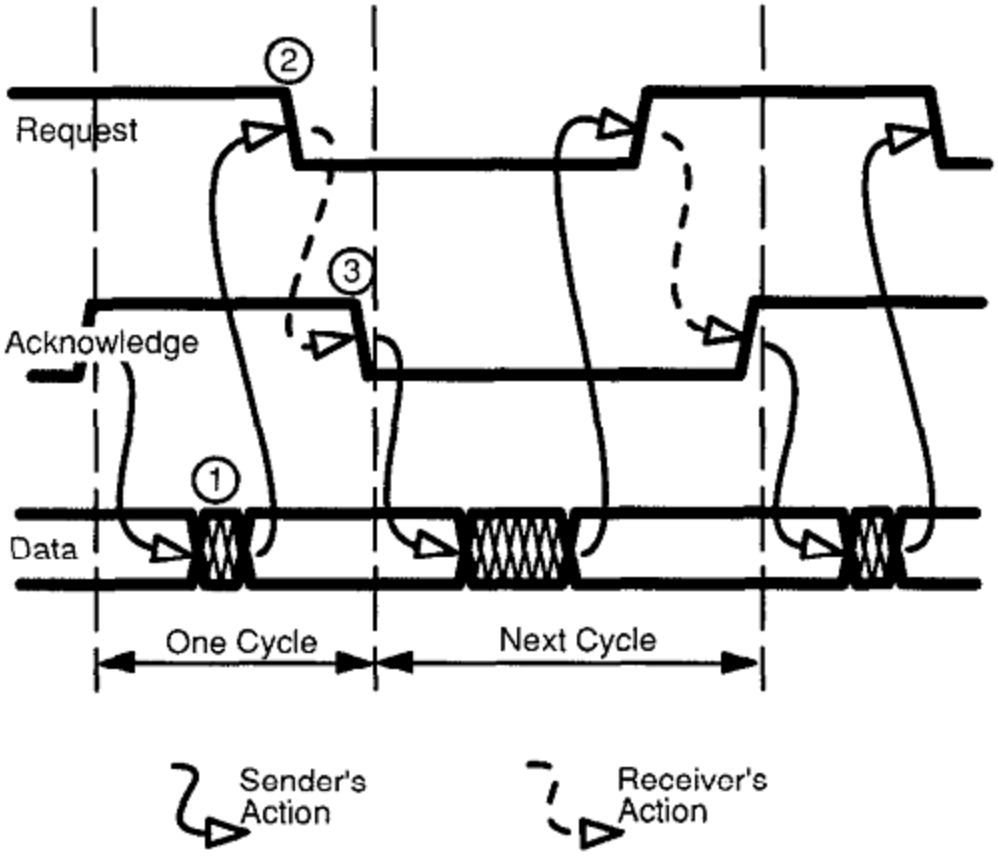

(a) Four-Phase

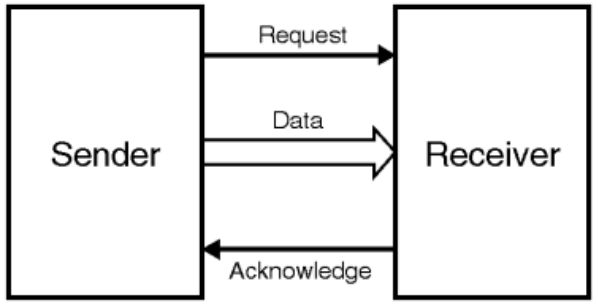
## Transition sensitive handshake




(b) Two-Phase

# TWO PHASE BUNDLE DATA CONVENTION

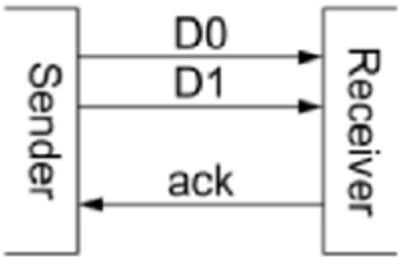


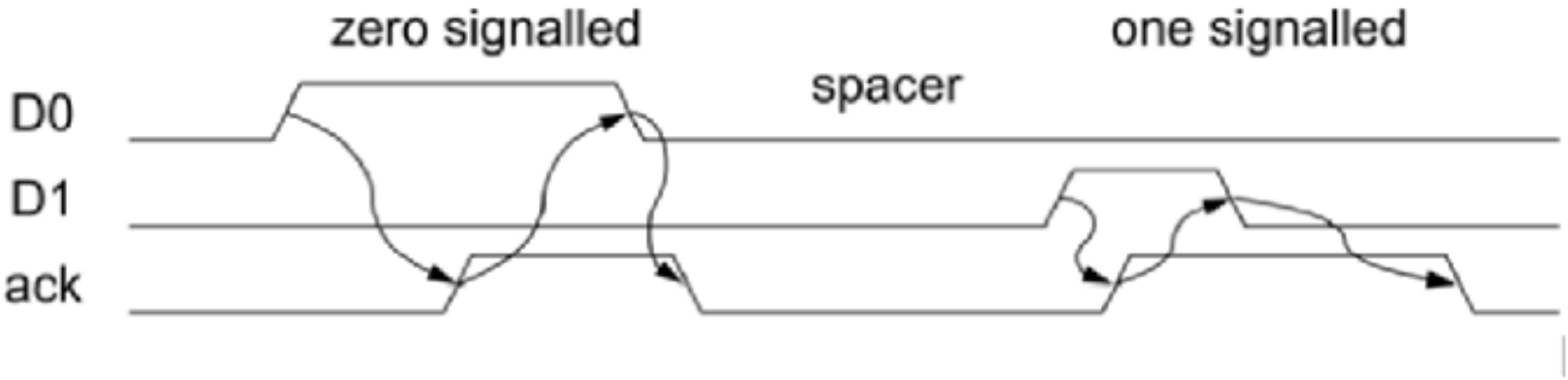




•Two phase bundledata interface

MICROPIPELINE HANDSHAKE PROTOCOL

# DUAL RAIL ENCODED HANDSHAKE




Dual Rail handshake using Level sensitive encoding

# How Asynchronous Systems are constructed to date?

- Extensive use pipelining
- Asynchronous pipelines are like a queue of traffic. Packets(vehicles) travelling serially along the pipeline(road)

No compulsion that a packet has to move every time!

- when there is a gap! Move forward ,otherwise wait!
- Packet position is uncertain , non local interactions are to be planned for ? e.g Register forwarding
  - **Register forwarding. Where are the results in the pipeline?**
- Synchronization is performed by handshakes. Whatever may be the choice between bundledata or the dual rail ,these are principles kept in mind

• The sender asserts a request
• When the receiver is Ready, it accepts data.
• After receiving data, receiver asserts acknowledgement

- **Assumptions in Synchronous design**

synchronous designer can make assumptions about the location of the data in the pipeline.

a copy of the data could be achieved at a particular time from a particular place.

NO HELP! For asynchronous designer. And they need to develop different mechanisms.

# FEATURES OF ASYNCHRONOUS DESIGN

- Elasticity
- Halt
- Data-dependent Timing
- Non-determinism

# 1. ELASTICITY

- Synchronous designers have to be well aware of the stalls and the their effects
- Stalls not only degrade performance but also force other parts of the system to adapt their behavior

⇨ This may result in excluding some desirable instructions from ISA because the implementation cost was very high.

• Of course Synchronous system will have to take care of worst case scenarios
• Slowest resource will result in global performance penalties effecting the on the system level.

⇨ • Asynchronous processors can take advantage here. Including an instruction Of the very high cost will not effect on the system level
• Pay more clock cycles only when those slow resources are in use !
• CMP instruction disappears after the ALU saving dummy WR and MEM Stages resulting in Power consumption reduction and performance increment

# 2. HALT

A system based on Low power CMOS technologies consumes very low power When halted.

Asynchronous processors may not be the fastest for doing something but are very good for doing nothing .

- **Asynchronous Systems**
- Inactivity Spreads rapidly.
- If one component stalls, interacting components also stall ! And that's the beauty of handshakes.
- HALT spreads naturally and full speed operation recommence from the deep-sleep instantly

  Suitable for the power sensitive embedded systems.
- **Synchronous Systems**
- In Synchronous Systems it's not easy to stop registers, clock and PLL
- Latency involved in awakening up the System from deepsleep can be very significant and discourages system halt.

# 3.Data dependent timing

- Some operations take longer than others to perform e.g an addition takes longer than bit-wise AND.
- This could be exploited in asynchronous systems.

Examples:

- Incrementer : average number of bit changes is almost 2.

  This could be very slow in worst case.

  Given a little amount of elasticity

# 4.Non Determinism

- A synchronous machine is totally deterministic .
- Given a particular System state, its future behavior could be exactly predicted.
- Asynchronous processors can also include some non determinism while still executing programs correctly

**So what the benefit of this non deterministic behavior?**

⇨ Scheduling of operations. Its sometime useful to allow the order in Which events happen to vary because of the unpredictable timing

⇨ Think about the prefetch depth.

It may save power consumption incase of a branch If the prefetched number of instructions (following branch) are less than the clocked equivalent of the same processor.

What is the cost of this non-deterministic behaviour?

# Problems with this non-deterministic behavior

- if an instruction turns out to be a branch then how to alter the pipeline flow at source?

1. Circulate tokens and avoid undeterministic prefetch !

   - So how to take advantage of non-determinism ?

2. Let the pipeline run freely but Allow only branches to redirect the flow

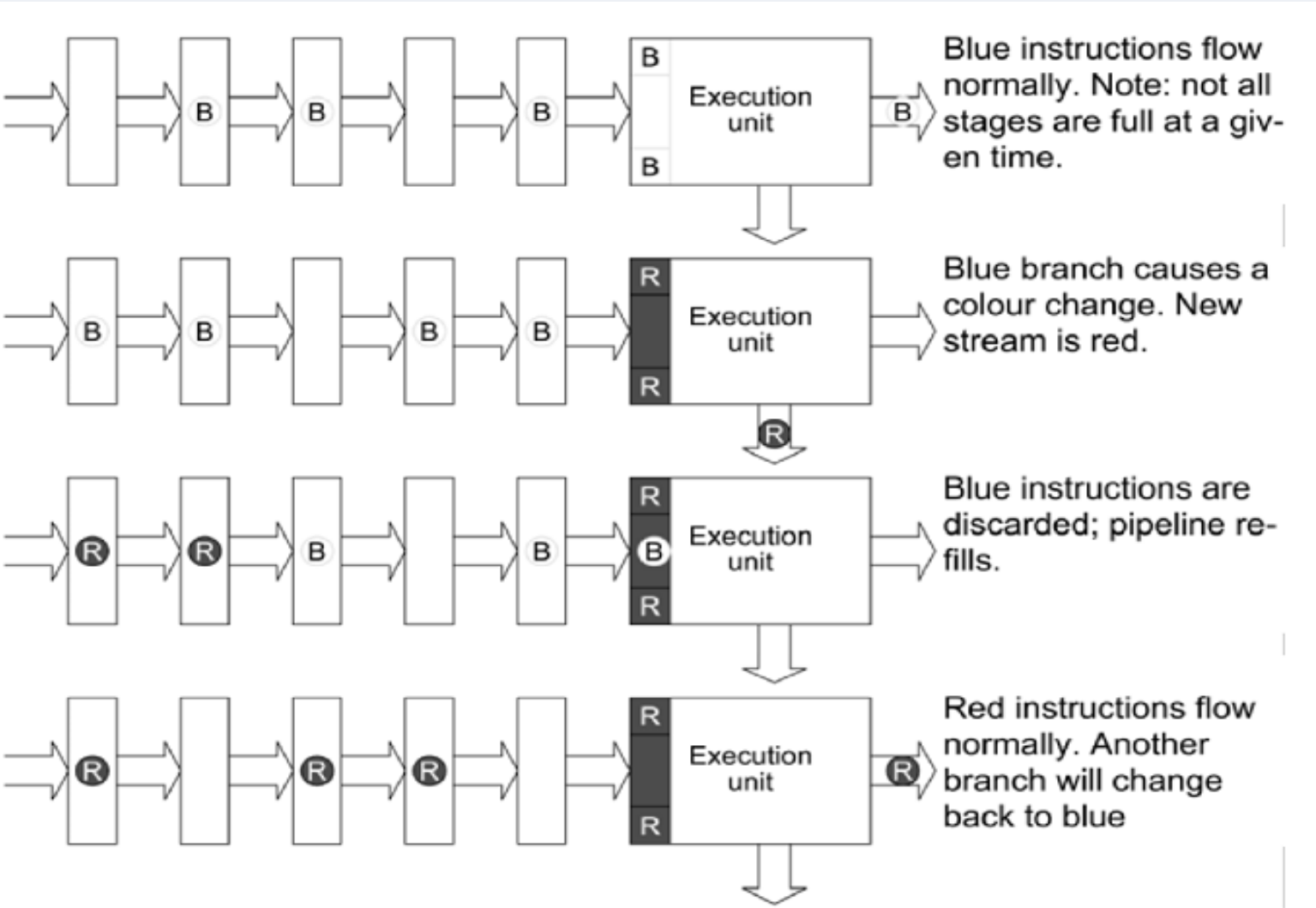
   - This approach results in undeterministic prefetch and introduces Fewer constraints.

     1. How to interrupt the flow reliably?
     2. How many prefetched instructions to discard due to unknown prefetch depth?

     - Arbiter can solve the first problem.
       Arbiter selects the order of access to a shared resource among asynchronous requests.

# Coloring Scheme to counter unknown prefetch depth

- Color the addresses at the prefetch unit and change the color when the flow changes. One bit was used for this scheme



- In amulet3 ,1 bit color scheme letter extended to 2 bits to allow late data aborts.

⇨ Imagine a data load is issued followed by a branch (taken)
• branch changes the color bit and prefetched color switches.
• data load(which usually takes more time) was going to commit
But the change of color simply discarded it!

- So introduce 1 more color bit to let instructions fetched before the branch commit .
- accurate redirection of the flow and accurate instructions discarded.

⇨ Non determinism and arbitration must be exploited carefully.

- It is possible to introduce cyclical dependencies resulting in a deadlock.

• suppose if arbiter decides to ignore the interruption for a while, and pipeline gets filled!
Solution: Move the taken branch of the main pipeline and let it flow by discarding prefetched operations.

- System verification:

•If designer want to explore all the states reachable by system to be sure Of the correctness of his design.
• each non-deterministic element multiplies up the number of possible Legal states.
• Simulation tools will not be able to explore all these states.

# Asynchronous Reorder buffer

- Its primary function is to

1. Accept results heading for retirement into registers and output them in a predetermined sequence.

   ⇨ This prevents WAW hazard

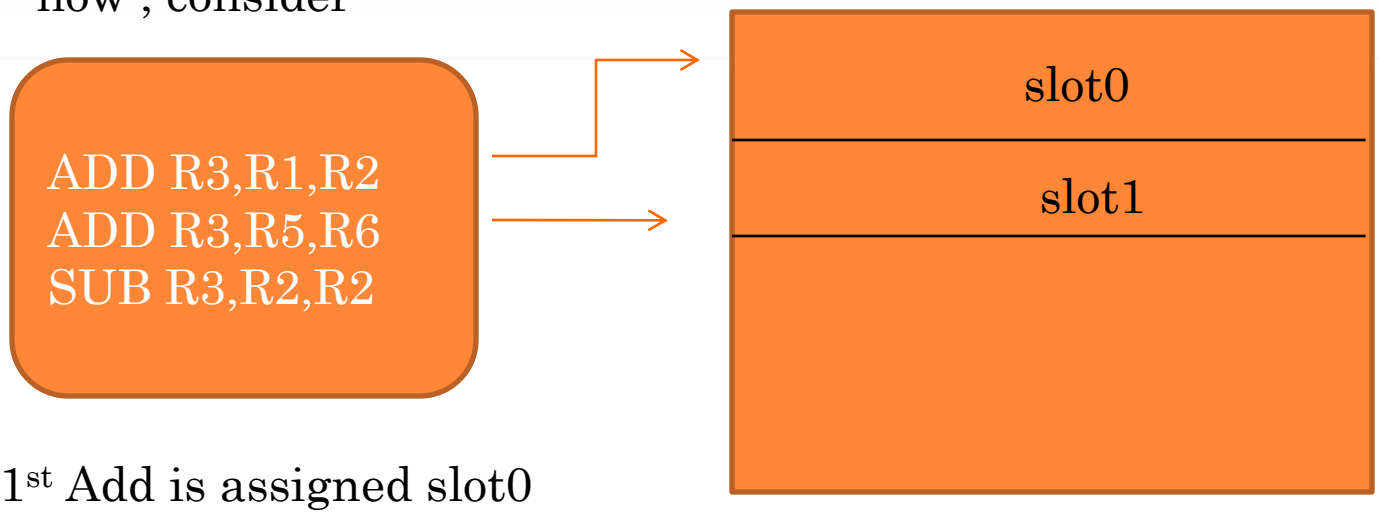
2. Forward values on demand to avoid Stalls whenever possible

**Asynchronous Implementation of ROB**

1. Process forwarding requests and allocate slots in the reorder buffer in the order they are to be readout.
2. After allocation the slot number is forwarded with the instruction packet for further re-use.
3. Now ,Some arbitrary times later ,it arrives at the reorder buffer.
4. Each slot in ROB has a unique number and only one packet could be assigned this number at a time.
5. It writes the results in the ROB according to the slot number.
6. Once written , ROB notes the arrival in the signals.
7. Now move around the ROB , write back the results and reset the arrival flag for reallocation and refilling.

## How ROB deals with the forwarding requests?

1. Decoder is responsible for issuing forwarding requests in a reverse chronological list of slots allocated to the required register
2. now , consider




1st Add is assigned slot0
2nd Add is assigned slot1

3. Now when SUB instruction is issued , decoder generates a request in {slot1,slot0}
4. Now first it checks in slot1 ,if value is available or not. If it is being computed then it waits for it.
5. If the data is invalid in slot1 , then it simply turns back to slot0 ,which will most probably result in invalid too.

- if both of the slots turns out to be invalid , it can then access an older value from the register bank
- Now imagine this

```
LDR    R0, [R1]
ADD    R1, R0, #1
```

- To check when data arives inside ROB ,status bit (same scheme like coloring) is used.\
- when LDR istruction is issued it is assigned slot0 as its destination.
- It turns to RED from BLUE ,when data arrives.
- Now ,decoding next instruction suggests that it want the result of previous instruction.
- Decoder will request forwarding {slot 0} and assign ADD instruction a slot in the ROB.
- Forwarding will take place only when the slot0 turns out to be RED( filled with the valid data)

⇒ As the ROB is cycled ,a few instruction latter ,slot0 will be reassigned.

```
SUB     R0, R0, #1    ;
```

- Imagine this SUB instruction .

1. First action is to forward existing R0 from slot0
2. Slot is then reassigned to SUB instruction .
3. This slot will not be over-written until the forwarding is complete.
4. When forwarding is done ,it will be again BLUE to wait for the results.
5. Again ,if there is some instruction after SUB to read R0 {slot0} will have to wait until this color gets changed to RED.

# ASYNCHRONOUS ROB HAZARDS

- It may issues more instructions to one slot like above (who are still blue and waiting for result to be written)
- it is prevented by throttle mechanism.

•A kind of FIFO ,through which the slot numbers are recycled.
•At RESET it holds number of tokens not more than the slots in buffer.
•AT decode , decoder collects a token from this FIFO and assigns to the instruction.
•If no token available it simply waits for a token.
•When a destination slot is written back to RF . Tokens are replenished to FIFO and could be taken again.

# Conclusion

- It is entirely practical to design a processor without a clock.
- Quality of such processors have improved with time. Some are commercially available
- So far , most of the efforts have been made to redesign already available architectures without a clock .
- Clockless and selftimed processors opens up horizons for a radical departure in the main stream architectures to new and exiting areas.
- CAD tools need to be worked on!

# Thanks =)