Asynchronous Circuit Design

Chris J. Myers

Lecture 9: Applications
Chapter 9

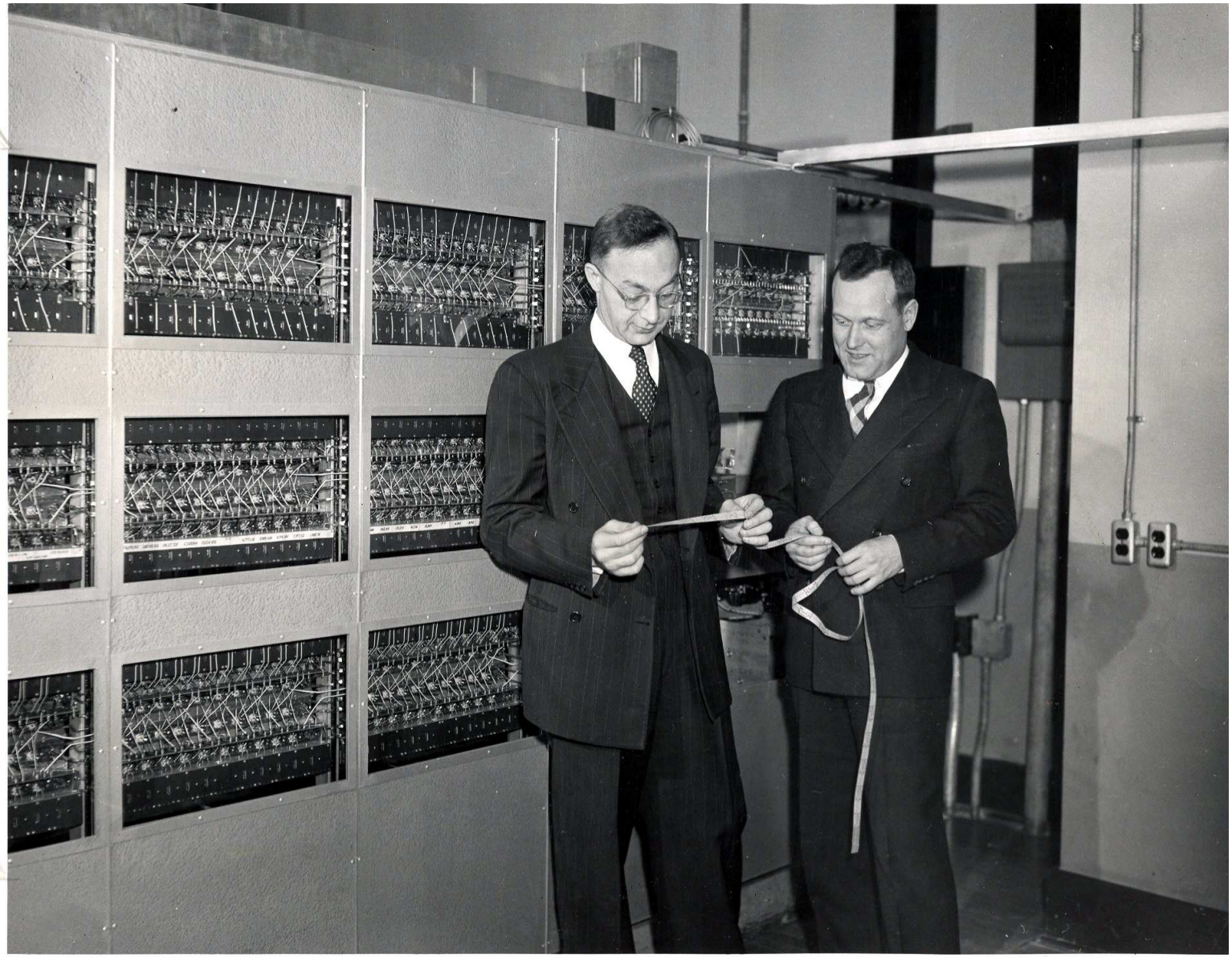
Overview

- A brief history of asynchronous circuit design
- Intel's RAPPID
- Performance analysis
- Testing asynchronous circuits
- The synchronization problem
- Arbitration
- The future of asynchronous circuit design

Early Mainframes

- In 50s and 60s, asynchronous design used in many mainframe computers, including ILLIAC and ILLIAC II designed at U. of Illinois and the Atlas and MU-5 designed at U. of Manchester.
- ILLIAC and ILLIAC II designed using speed-independent design by Muller and his colleagues.
- ILLIAC, completed in 1952, was 10 feet long, 2 feet wide, 8 $\frac{1}{2}$ feet high, contained 2800 vacuum tubes, and weighed 5 tons.

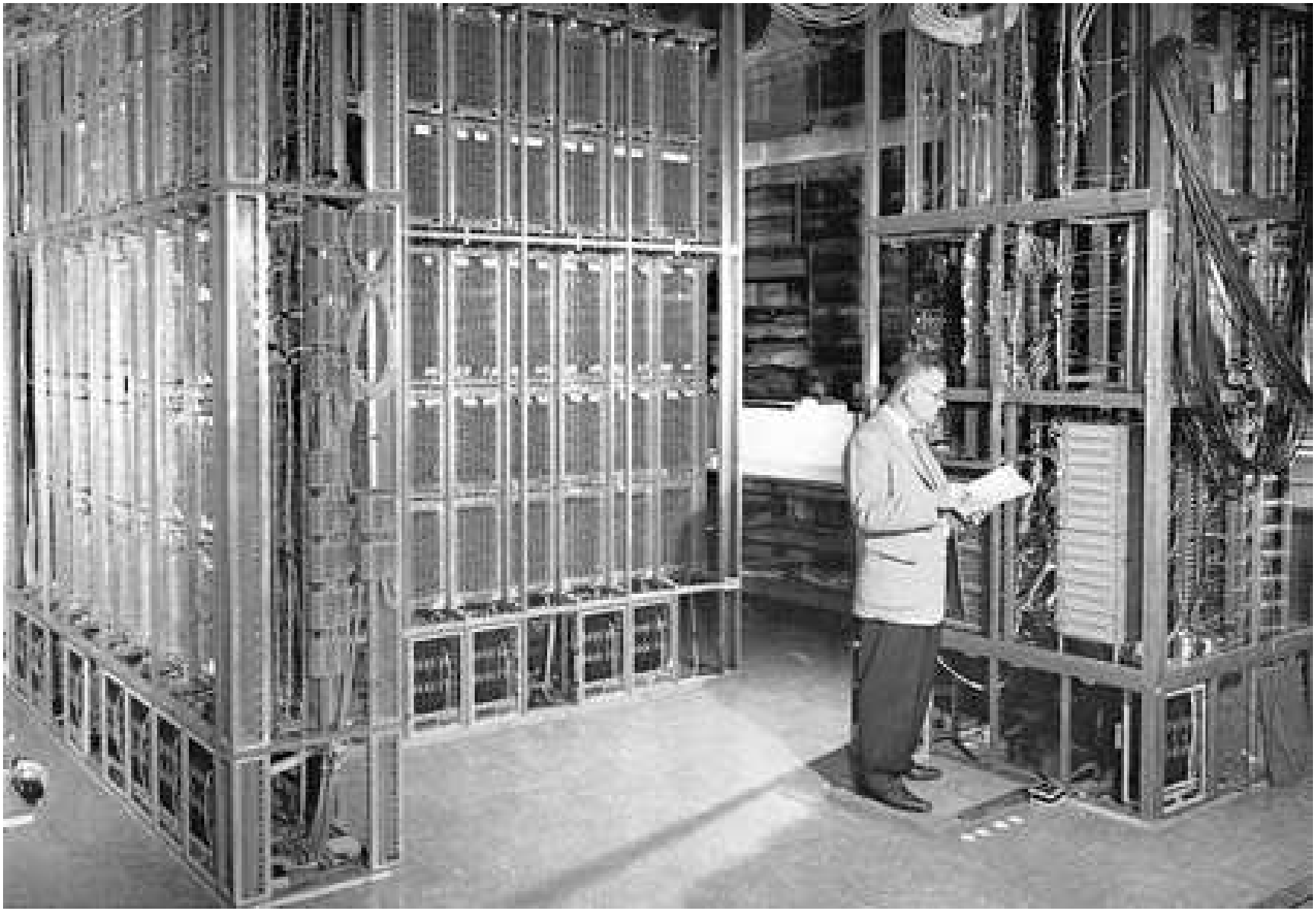
ILLIAC



ILLIAC II

- ILLIAC II completed in 1962 was 100 times faster than its predecessor.
- Had 55,000 transistors and performed a FP multiply in 6.3 μs.
- Used three concurrently operating controls: an arithmetic control, interplay control, and Advanced Control.
- Three controls are largely asynchronous and speed-independent.
- SI design used to increase reliability and ease of maintenance.
- Controls collected *reply* signals to indicate that all operations for current step are complete before going on to next step.
- Arithmetic unit was not speed-independent, as it would increase its complexity and cost while decreasing its speed.
- Electromechanical peripheral devices were also not speed-independent, since they were inherently synchronous.
- Used until 1967, and found prime, $2^{11213} - 1$ (over 3000 digits).

ILLIAC II

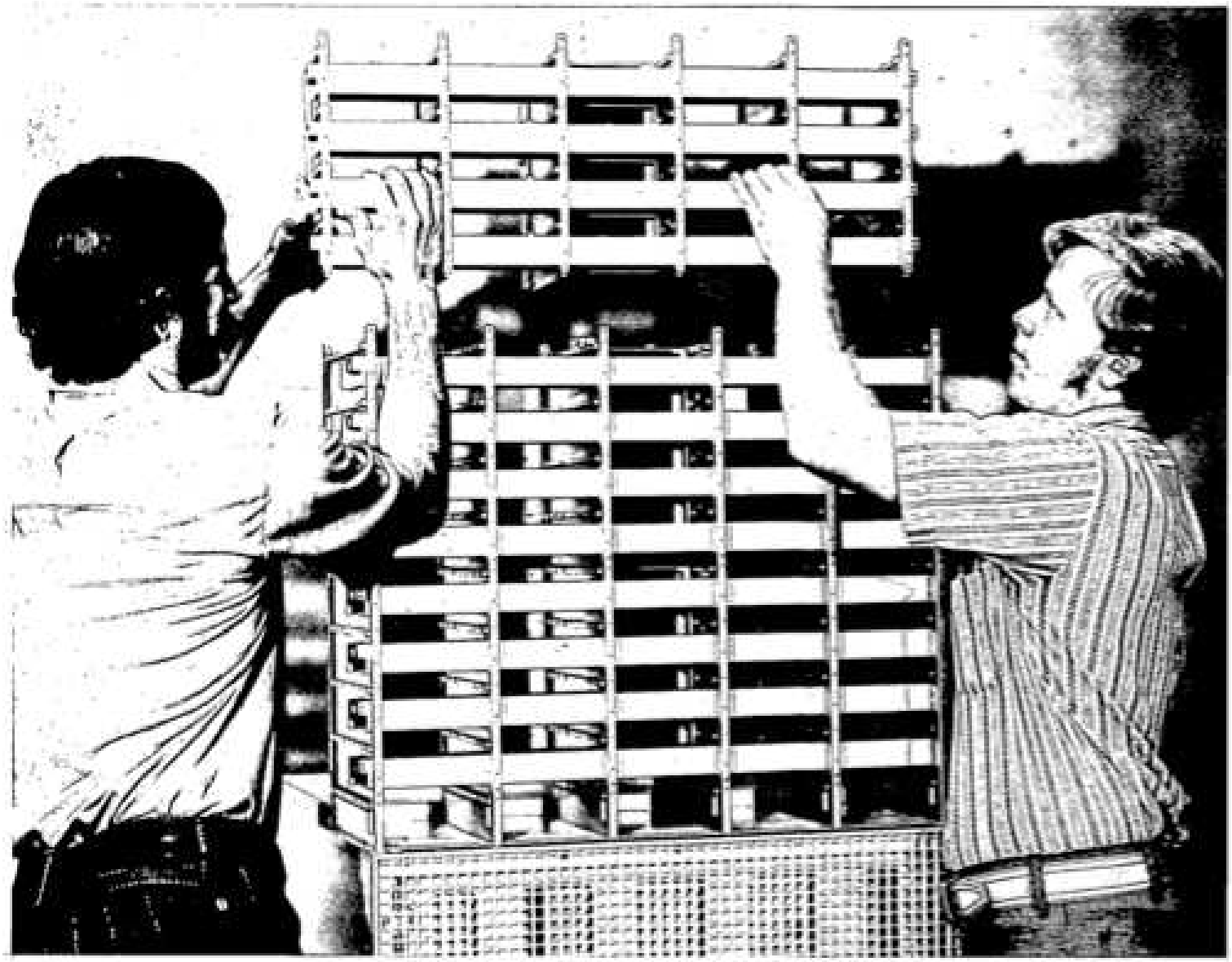


ILLIAC II Control Panel

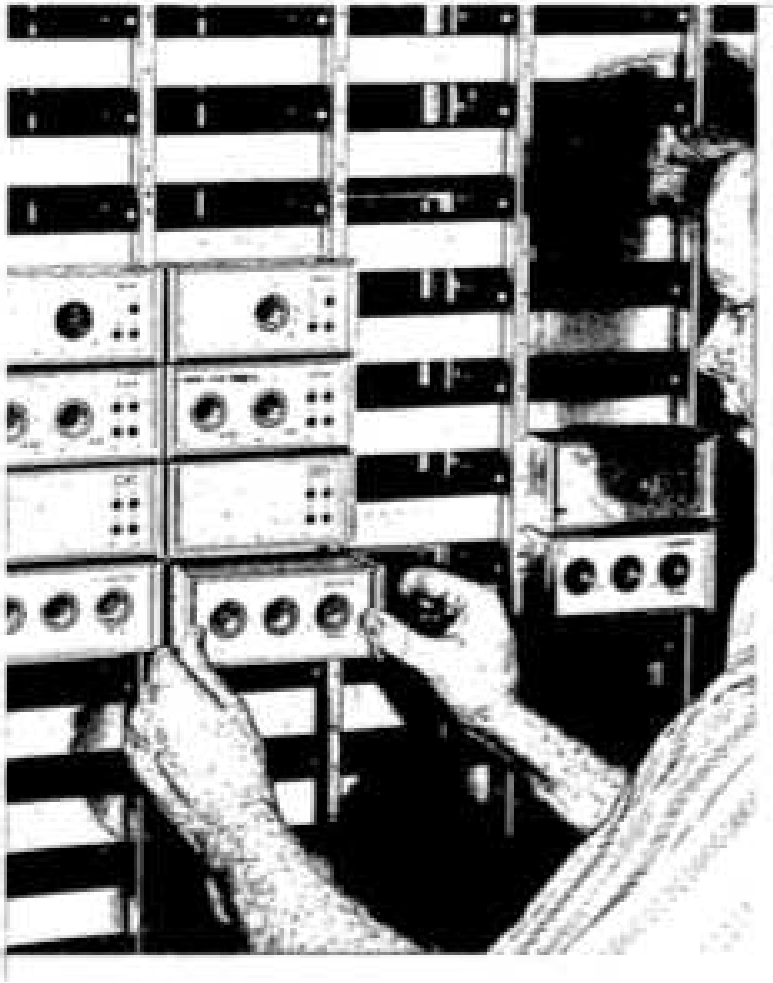
Macromodules

- Developed in 60s and 70s at Washington University in St. Louis.
- Macromodules are “building blocks ... from which it is possible for the electronically-naive to construct arbitrarily large and complex computers that work.”
- Asynchronous to allow easy interconnection and to allow designers to worry only about logical and not electrical problems.
- Used to build macromodular computer systems by placing in a rack and interconnecting with wires.
- Wires carry data signals as well as a bundled data control signal.
- Wires also for control to sequence operations.
- Developed directly from a flowchart description of an algorithm.
- Many computing engines designed using macromodules.

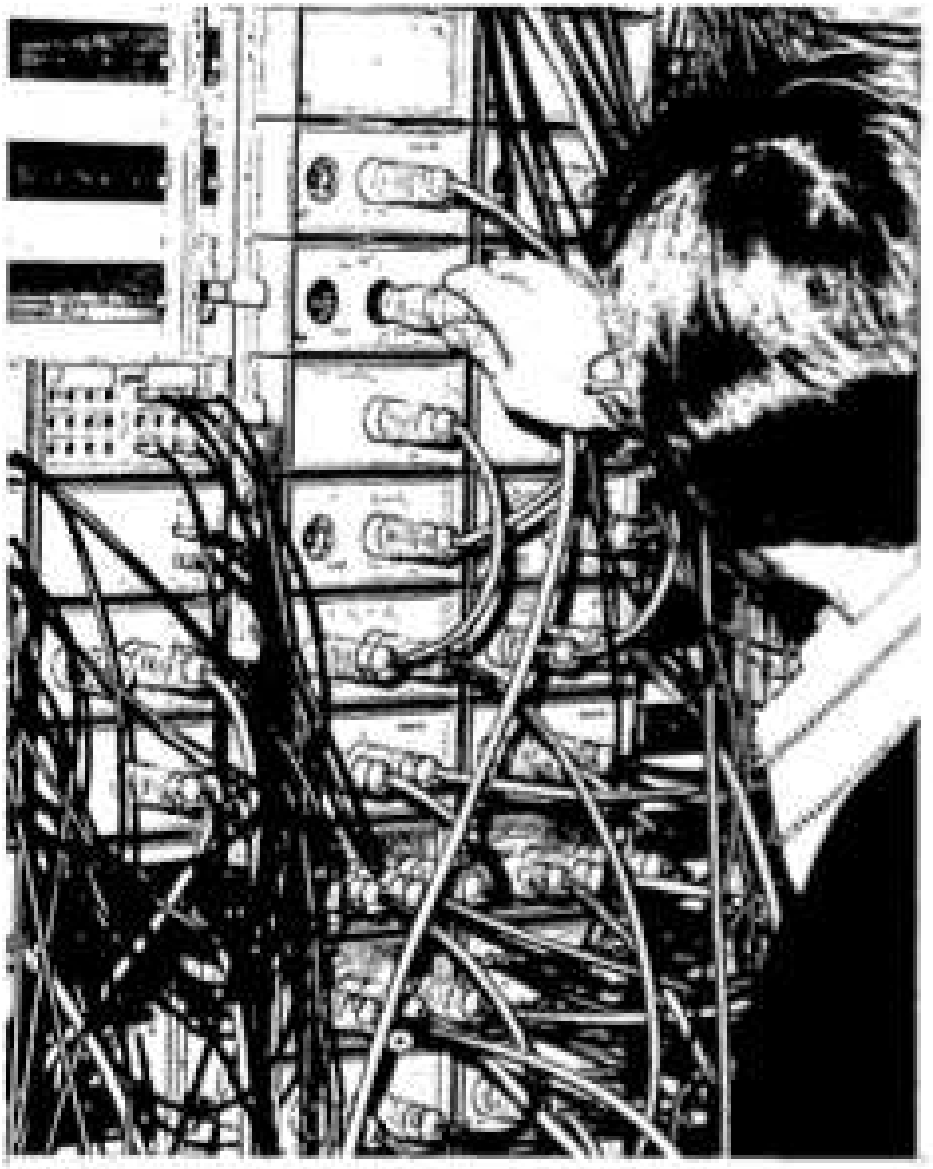
Macromodules



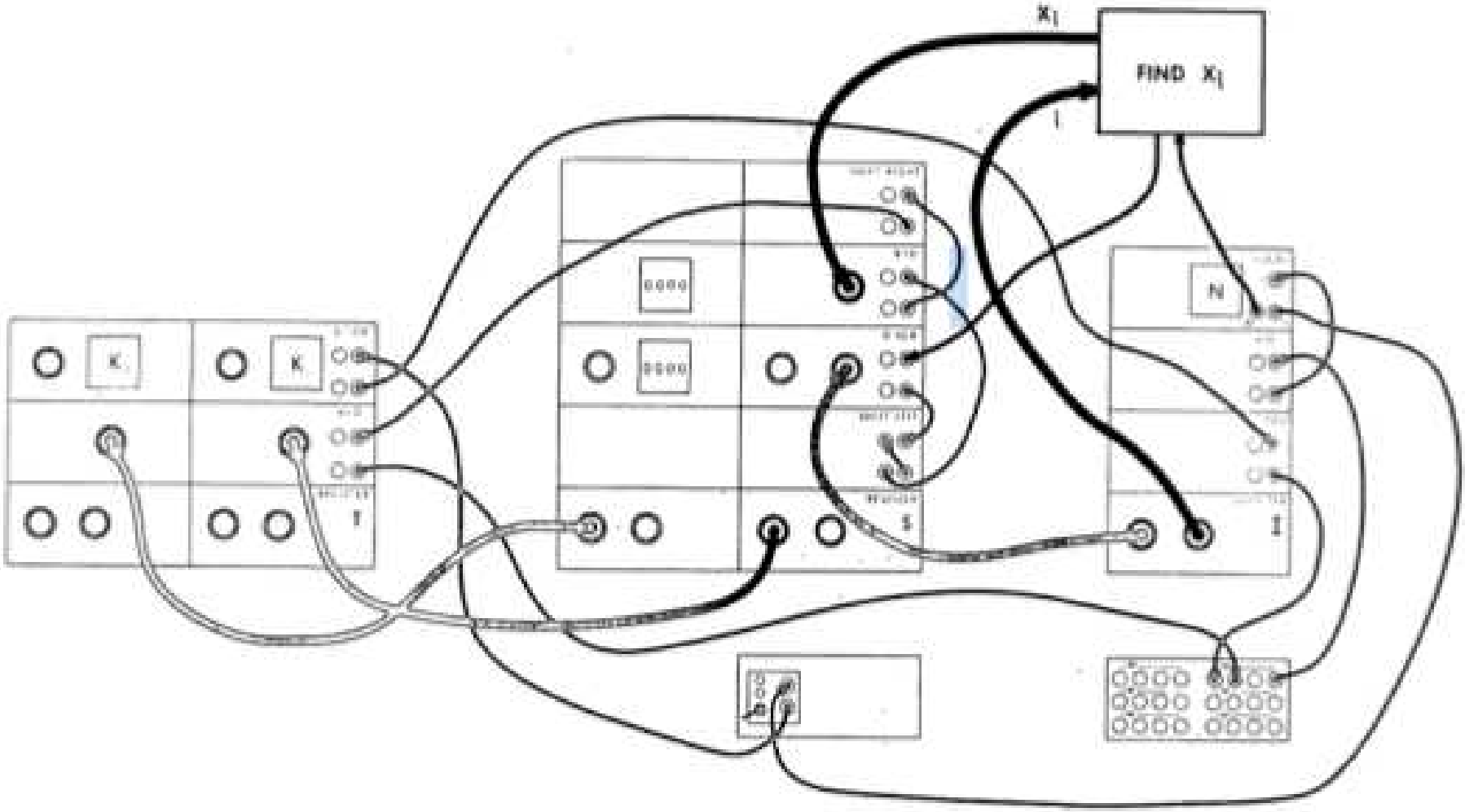
Macromodules



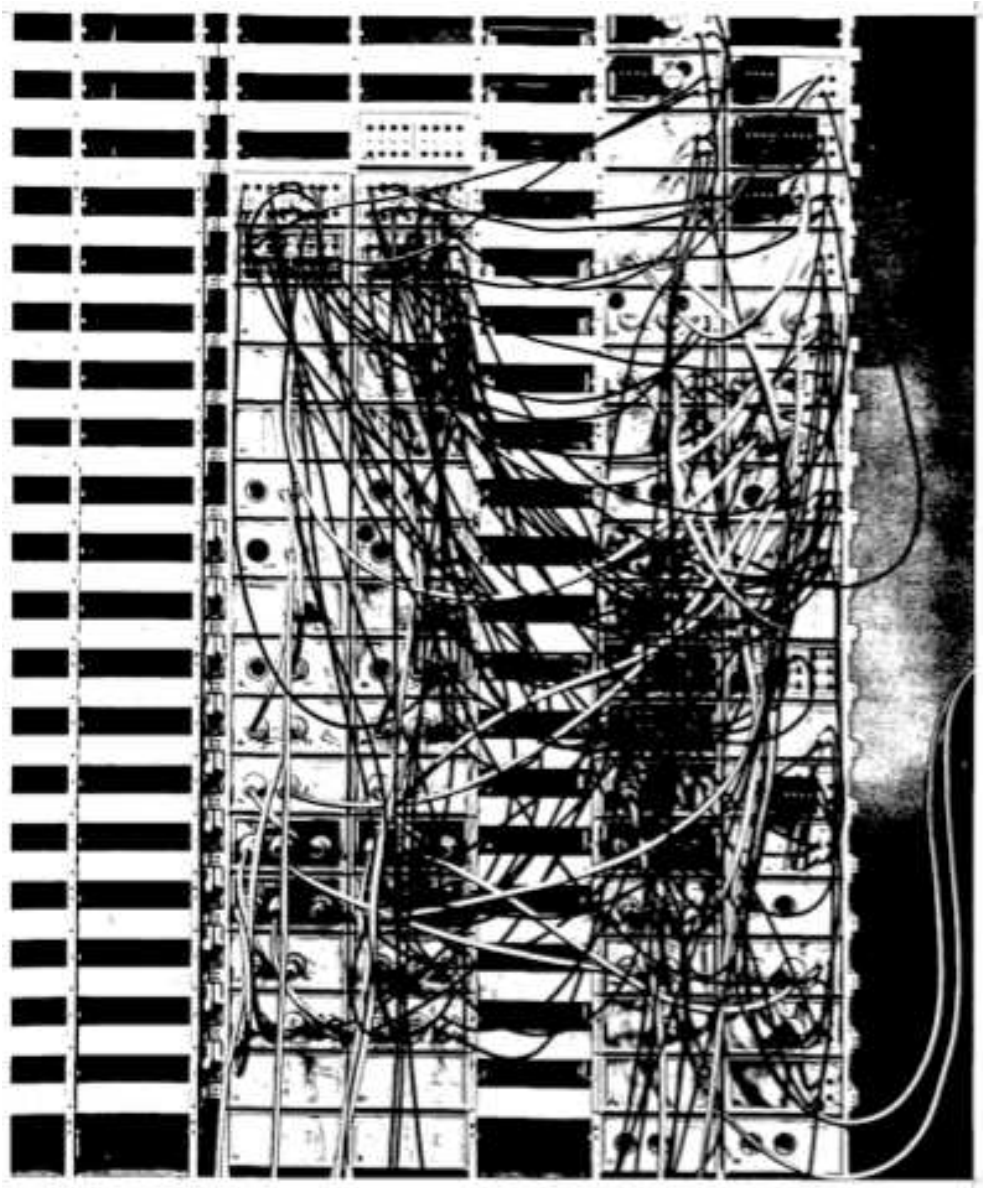
Macromodules



Macromodules



Macromodules



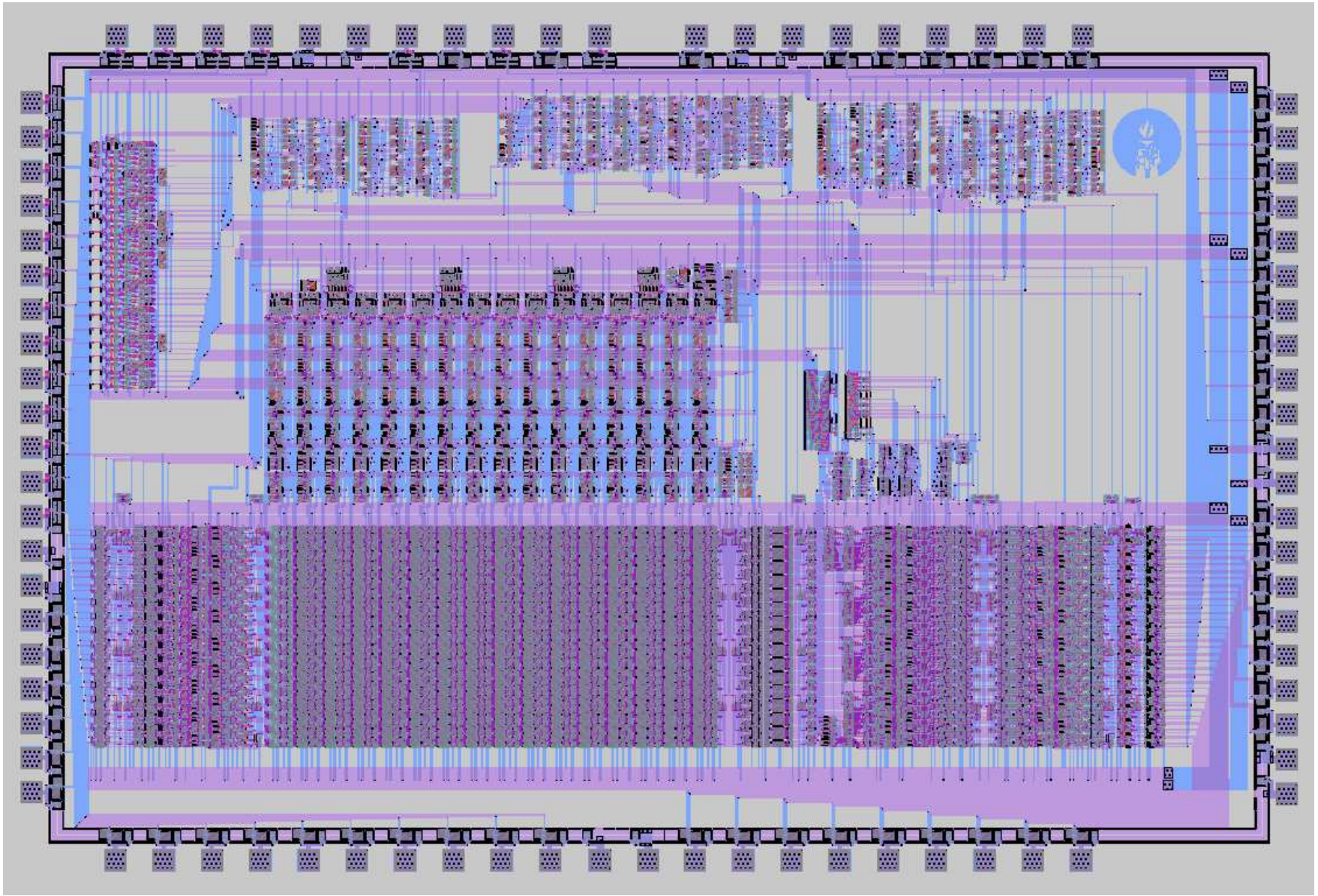
Data Driven Machines

- Rather than single PC, flow of data controls operation speed.
- In 70s, first operational dataflow computer, DDM-1, designed at U. of Utah and first commercial graphics system designed at Evans and Sutherland.
- In the 80s, Matsushita, Sanyo, Sharp, and Mitsubishi designed several *data-driven processors*.
- Most recently, a data-driven media processor (DDMP) has been designed at Sharp capable of 2400 million signal processing operations per second while consuming only 1.32 W.
- Videonics has utilized DDMP design in a high-speed video DSP.
- Asynchronous design of DDMP is cited to have simplified the board layout and reduced RF interference.

Caltech's Asynchronous Microprocessors

- In 1989, designed first fully asynchronous microprocessor.
- Processor has a 16-bit datapath with 16- and 32-bit instructions.
- Has twelve 16-bit registers, 4 buses, an ALU, and 2 adders.
- Consists of 20,000 transistors, fabricated in 2 μm and 1.6 μm, and took five people only five months to design.
- 2 μm version could perform 12 million ALU instructions per second, while the 1.6 μm version could perform 18 million.
- Chips operate with VDD from 0.35 to 7 V, and achieve almost double the performance when cooled in liquid nitrogen.
- Design is entirely *quasi-delay insensitive* (QDI).
- Design was derived from a high-level channel description using program transformations to introduce ideas like pipelining.

Caltech's Asynchronous Microprocessors



Caltech's Asynchronous Microprocessors (cont)

- Also designed first asynchronous GaAs microprocessor.
- This processor ran at 100 MIPS while consuming 2 W.
- Designed an asynchronous MIPS R3000 microprocessor.
- Design fabricated in 0.6 μm CMOS, and it uses 2 million transistors, of which 1.25 million are in its caches.
- Measured performance ranged from 60 MIPS and 220 mW at 1.5 V and 25°C to 180 MIPS and 4 W at 3.3 V and 25°C.
- Running Dhrystone, the chip achieved about 185 MHz at 3.3 V.

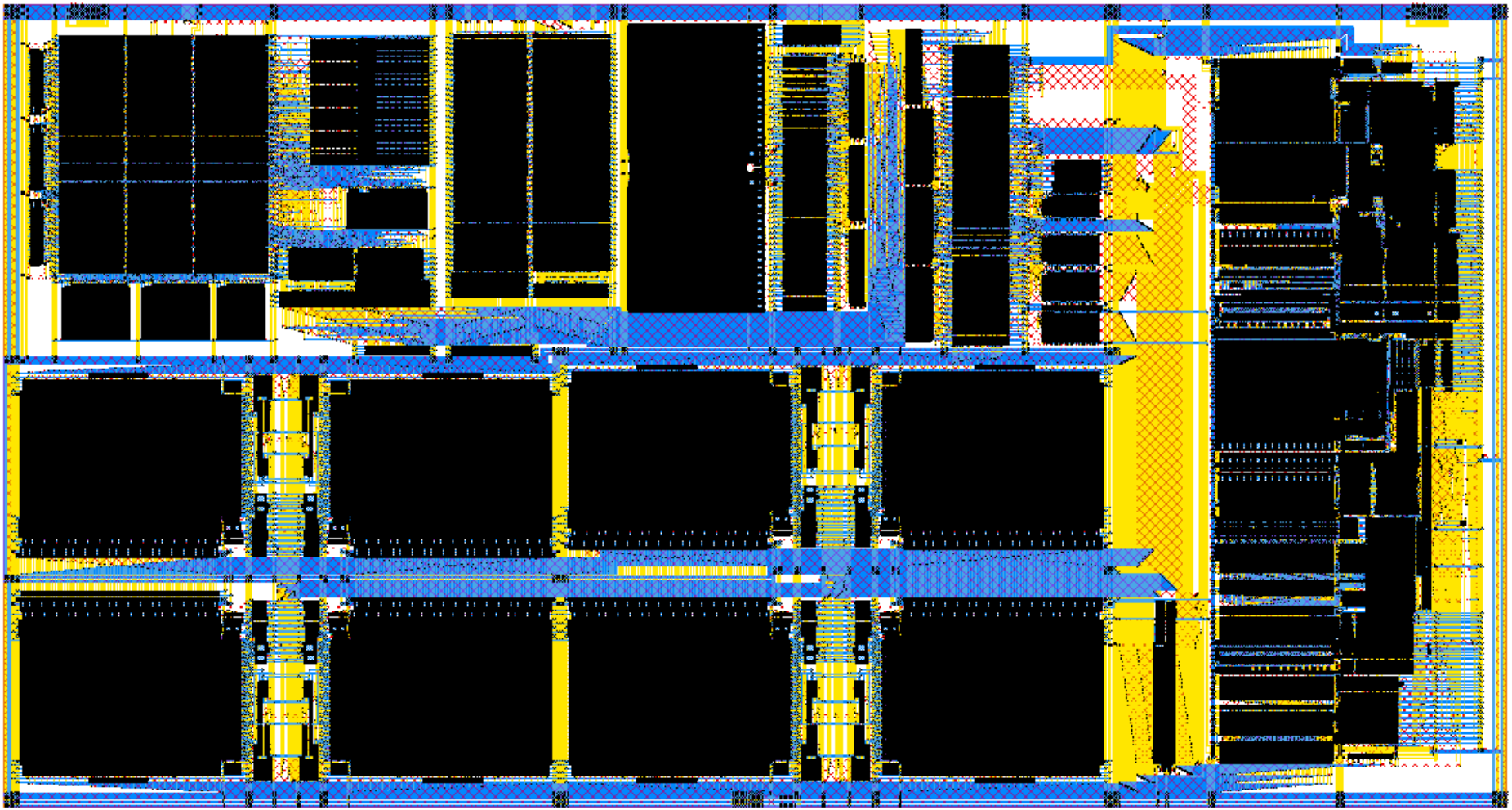
AMULET

- AMULET1 completed in 1994 at U. of Manchester was the first asynchronous processor to be code-compatible with an existing synchronous processor, the ARM.
- Design style followed Sutherland's two-phase micropipeline idea.
- Chip fabricated in a 1 μm and a 0.7 μm process.
- Performance was measured for the 1 μm part from 3.5 to 6 V completing between 15 and 25 thousand Dhrystones per second.
- MIPS/watt value was also measured to be from 175 down to 50.
- Chip operates correctly between -50°C and 120°C.

AMULET (cont)

- AMULET2e, targeting embedded systems, contained an AMULET2 core with a cache/RAM, a memory interface, and other control functions on chip.
- This design used four-phase bundled-data style.
- Design was fabricated in 0.5 μm CMOS, and its measured performance at 3.3 V was 74 kDhrystones, which is roughly equivalent to 42 MIPS and consumed 150 mW.
- The radio-frequency emission spectrum or EMC was shown to be significantly spread as compared with a clocked system.
- When processor enters “halt” mode, power is under 0.1 mW.
- AMULET3 has incorporated ARM thumb code and new architectural features to improve performance.

AMULET 3 Microprocessor



SUN Pipelines

- In 1994, group at SUN suggested replicating synchronous architecture may not be the best for asynchronous design.
- Suggested a radically different architecture, the *counterflow pipeline* in which instructions are injected up the pipeline while contents of registers are injected down.
- When instruction meets register values, computes a result.
- Key to the performance of such a design are circuits to move data very quickly.
- This group has designed several very fast FIFO circuits.
- Test chips fabricated in 0.6 μm have a maximum throughput of between 1.1 and 1.7 Giga data items per second.

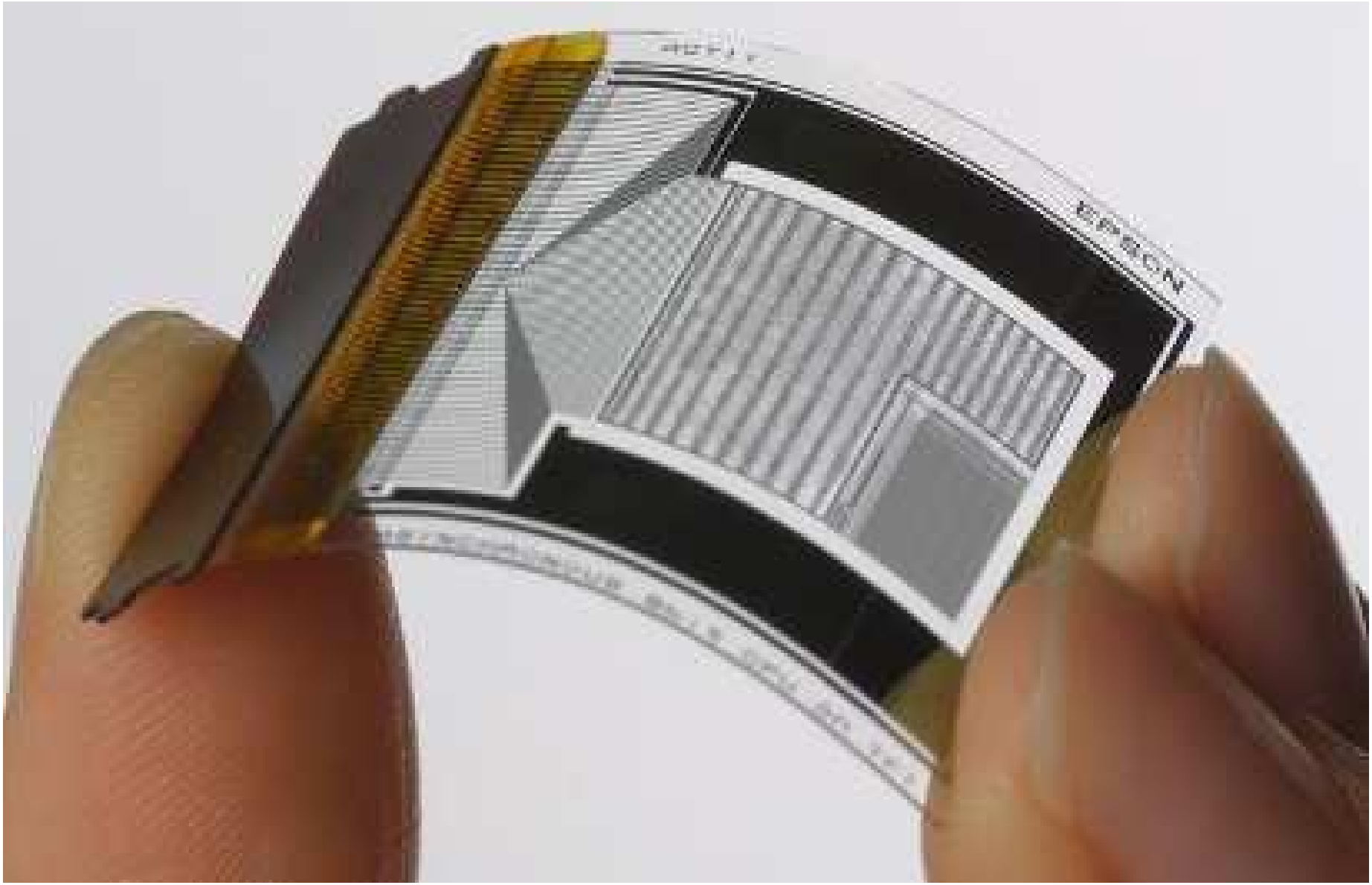
Designs at Philips Research Laboratories

- Designed many asynchronous designs targeting low power.
- Developed design procedure from specification in TANGRAM to a chip, and applied to several commercially interesting designs.
- In 1994, produced an error corrector chip for a DCC player that consumed only 10 mW at 5 V, $\frac{1}{5}$ of synchronous counterpart.
- Design also required only 20 percent more area.
- In 1997, designed standby circuits for a pager decoder which uses 4 times less power while 40% larger than synchronous design.
- Results in a 37 percent decrease in power for entire pager.
- In 1998, this group designed an 80C51 microcontroller.
- Chip in 0.5 μm is 3 to 4 times more power efficient than synchronous counterpart, consuming only 9 mW at 4 MIPS.

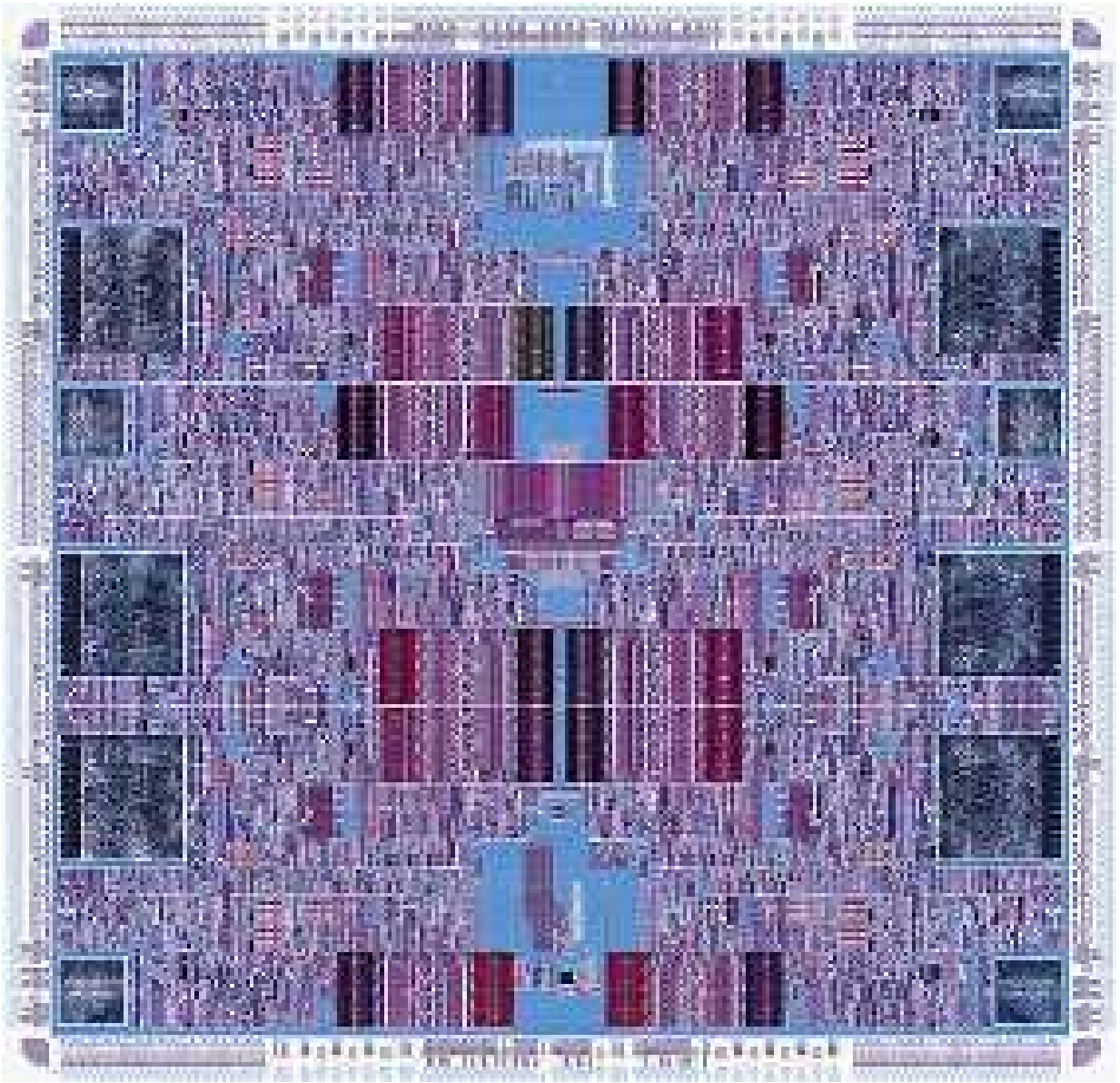
Designs at Philips Research Laboratories

- Most notable accomplishment is a fully asynchronous pager sold by Philips using standby circuits and 80C51 microcontroller.
- Major reason Philips uses asynchronous pager is it has a more evenly spread emission spectrum over the frequency range.
- Interference at clock frequency and harmonics requires digital circuitry in synchronous design to be shut off as message arrives.
- Spread-spectrum emission for asynchronous design allows digital circuitry to remain active as the message is received.
- Permits pager to be capable of being universal in that it can accept all three of the international pager standards.

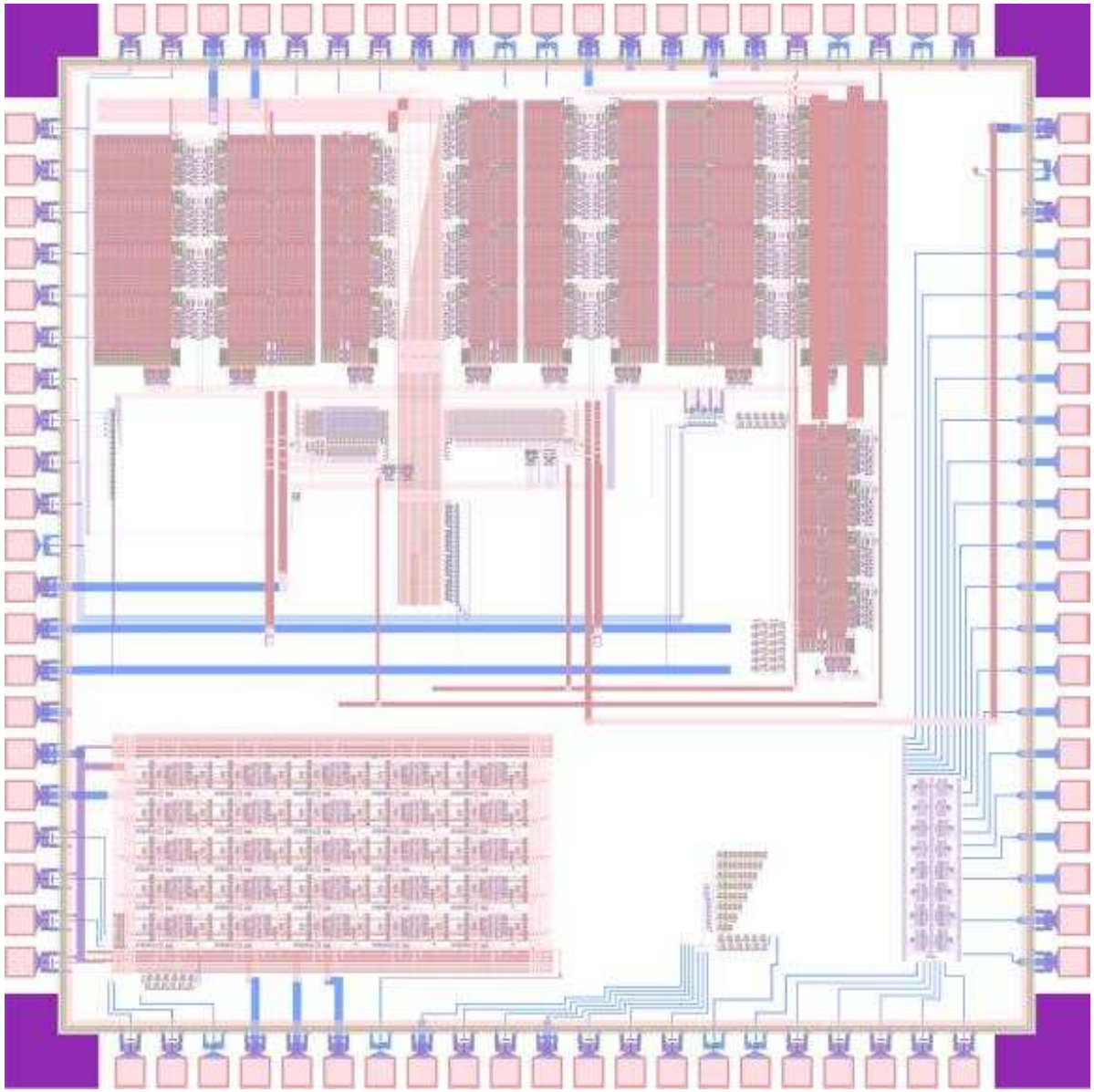
Epson's Flexible 8-bit Asynchronous Microcontroller



Fulcrum's 1GHz Processor using Asynchronous Circuits



Cornel's Asynchronous FPGAs (Achronix)



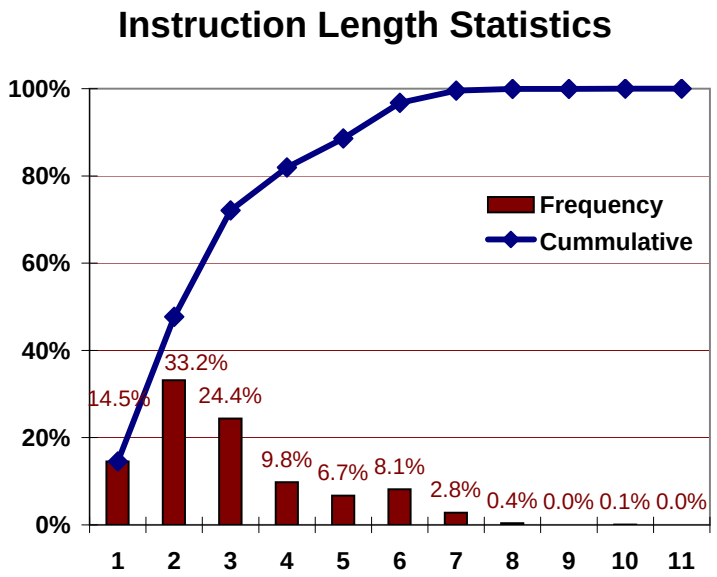
An Asynchronous Instruction-Length Decoder

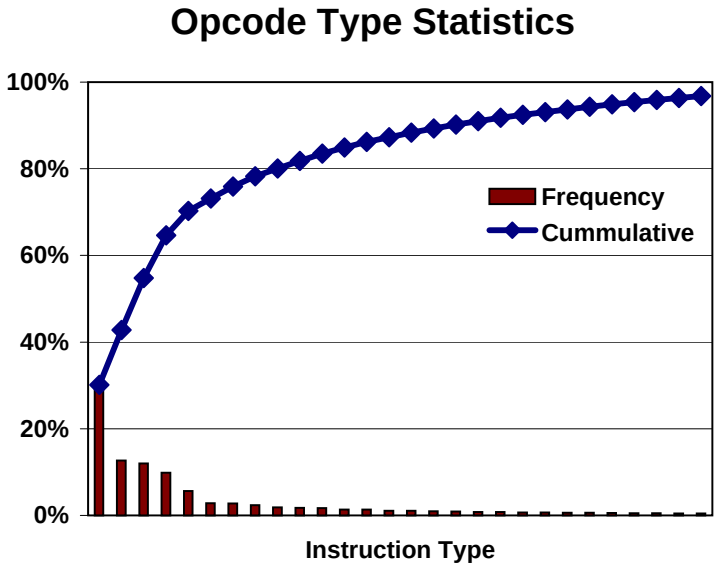
- RAPPID (Revolving Asynchronous Pentium Processor Instruction Decoder) is a fully asynchronous instruction-length decoder for the complete PentiumII 32-bit MMX instruction set.
- RAPPID project conducted at Intel between 1995 and 1999.
- Achieved 3 fold improvement in speed and 2 fold improvement in power compared with existing synchronous design.

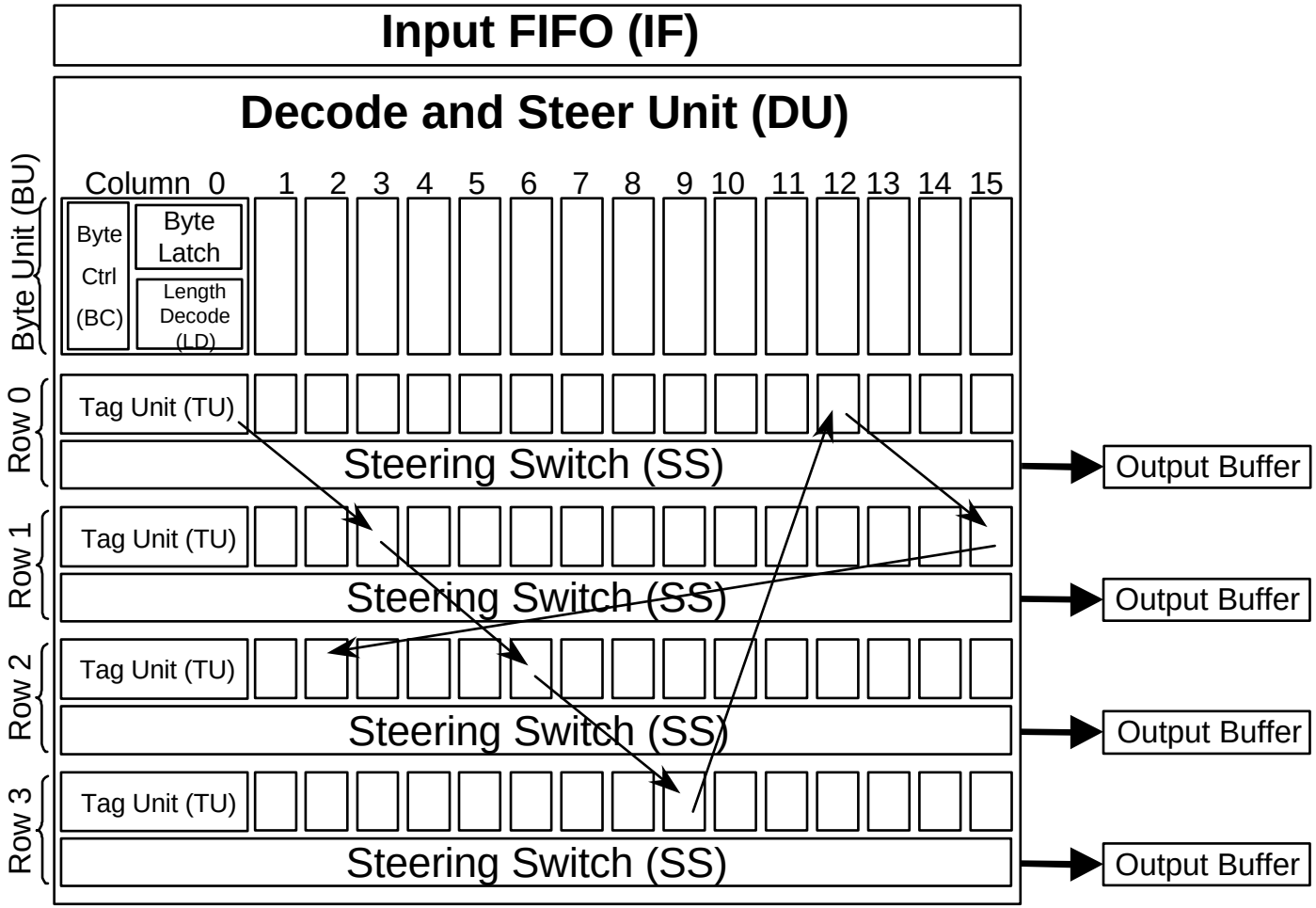
x86 Instruction-Length Decoding

- Each instruction can be from 1 to 15 bytes long.
- To allow concurrent execution of instructions, necessary to rapidly determine positions of each instruction in a cache line.
- It was the critical bottleneck in this architecture.
- A partial list of rules to determine instruction length:
  - Opcode can be 1 or 2 bytes.
  - Opcode determines presence of the ModR/M byte.
  - ModR/M determines presence of the SIB byte.
  - ModR/M and SIB set length of displacement field.
  - Opcode determines length of immediate field.
  - Instructions may be preceded by as many as 15 prefix bytes.
  - A prefix may change the length of an instruction.
  - The maximum instruction length is 15 bytes.

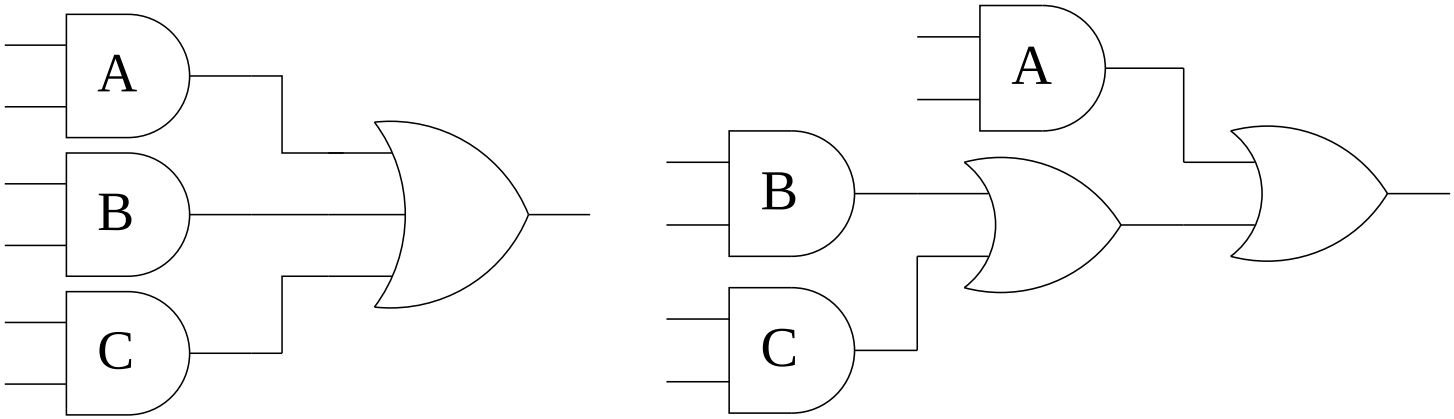
RAPPID Data


Instruction Length Statistics
100%
80%
60%
40%
20%
0%
Frequency
Cummulative
14.5%
33.2%
24.4%
9.8%
6.7%
8.1%
2.8%
0.4%
0.0%
0.1%
0.0%
1
2
3
4
5
6
7
8
9
10
11

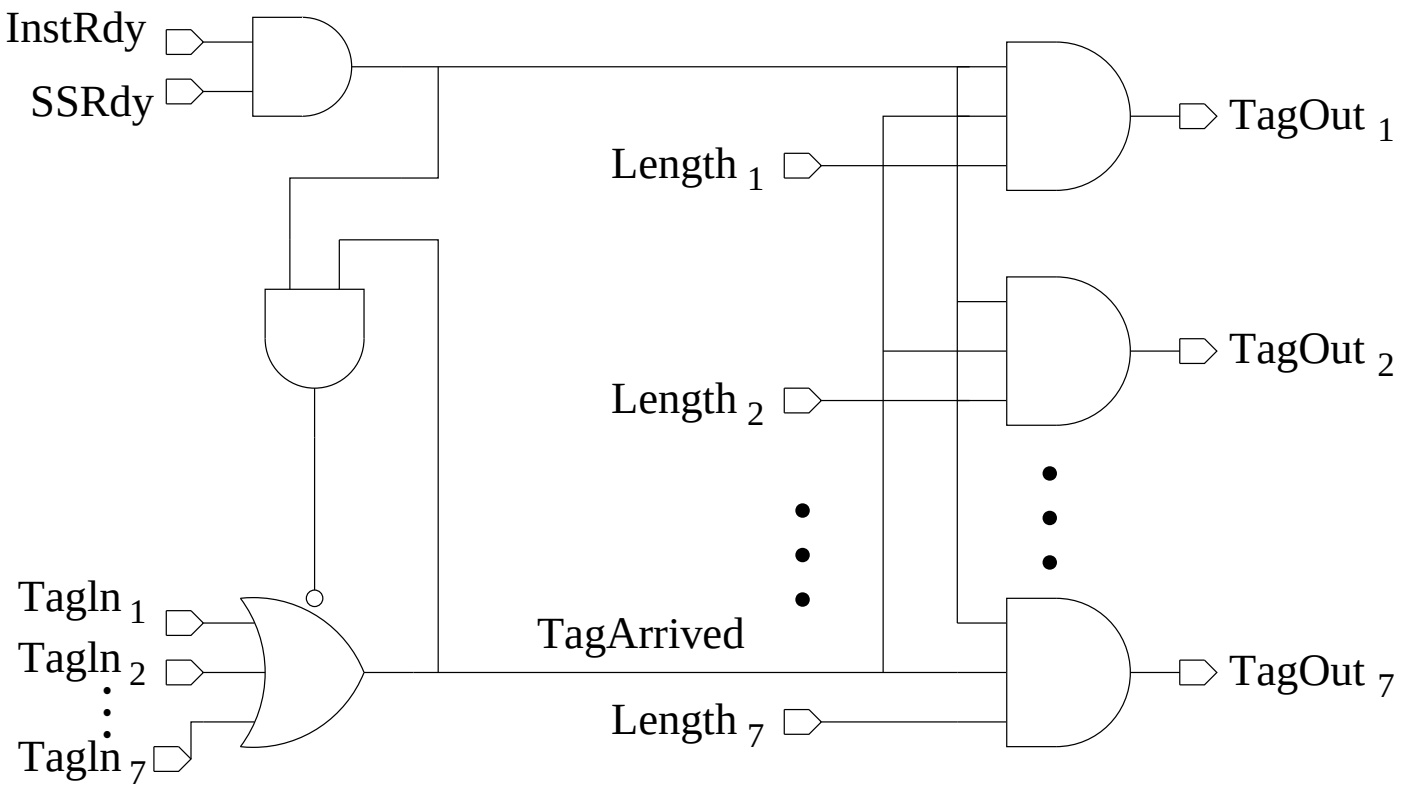

Opcode Type Statistics
100%
80%
60%
40%
20%
0%
Frequency
Cummulative
Instruction Type


Input FIFO (IF)
Decode and Steer Unit (DU)
Column 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Byte Unit (BU)
Byte Ctrl (BC)
Byte Latch
Length Decode (LD)
Row 0
Tag Unit (TU)
Steering Switch (SS)
Row 1
Tag Unit (TU)
Steering Switch (SS)
Row 2
Tag Unit (TU)
Steering Switch (SS)
Row 3
Tag Unit (TU)
Steering Switch (SS)
Output Buffer
Output Buffer
Output Buffer
Output Buffer

Balanced Versus Unbalanced Logic



Tag Unit Circuit



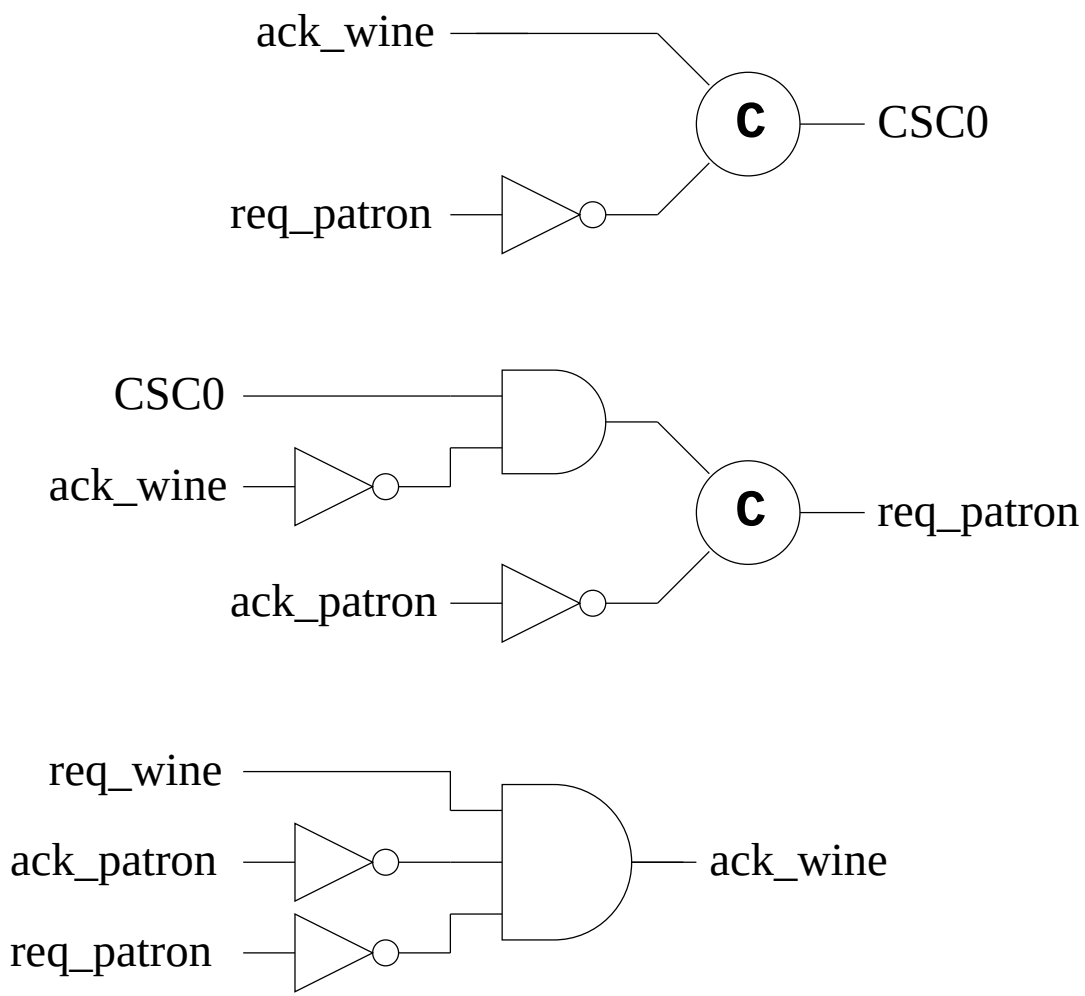
RAPPID Test Results

- Test chip fabricated in May 1998 using a 0.25 μm process.
- Capable of decoding and steering instructions at a rate of 2.5 to 4.5 instructions per ns.
- Fastest synchronous 3-issue product in same fabrication process clocked at 400 MHz is capable of only 1.2 instructions per ns.
- Chip operated correctly between 1.0 and 2.5 V while synchronous design could only tolerate about 1.9 to 2.1 V.
- Consumes only $\frac{1}{2}$ of energy of clocked design.
- Found to achieve these gains with only a 22 percent area penalty.

Performance Analysis

- For asynchronous design, cannot simply find critical path delay or count number of clock cycles per operation.
- Worst-case analysis may be quite pessimistic, as goal is to achieve high rates of performance on average.
- Must take a probabilistic approach to performance analysis.
- Need to extend model to include a distribution function for each delay.
- Simple approach is assume delay is uniform in its range.
- Another possibility is to use a truncated Gaussian.
- Most direct approach to determine performance analysis is a Monte Carlo simulation.

Wine Shop Example: Timed Circuit



Timing Assumptions

Assumption	Delay
Winery delays	2 to 3 minutes
Patron responds	5 to ∞ minutes
Patron resets	2 to 3 minutes
Inverter delay	0 to 6 seconds
AND gate delay	6 to 12 seconds
OR gate delay	6 to 12 seconds
C-element delay	12 to 18 seconds

Synchronous worst-case cycle time is 18.3 minutes

Asynchronous average-case cycle time is 14.2 minutes

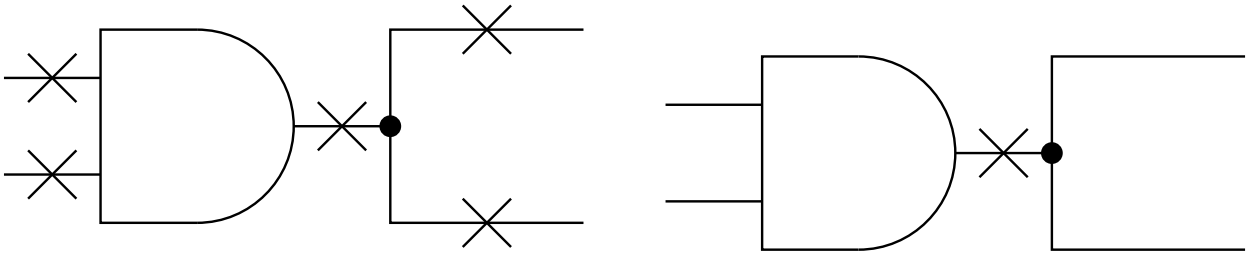
Testing Asynchronous Circuits (Bad News)

- Once chip is fabricated, must test it to determine presence of manufacturing defects, or *faults*, before delivering to consumer.
- In asynchronous circuits, there is no global clock which can be used to single step the design through a sequence of steps.
- Asynchronous circuits have more state holding elements, which increases overhead to apply and examine test vectors.
- Huffman circuits use redundant circuitry to remove hazards which tends to hide some faults, making them untestable.
- Asynchronous circuits may fail due to glitches caused by *delay faults*, which are difficult to detect.

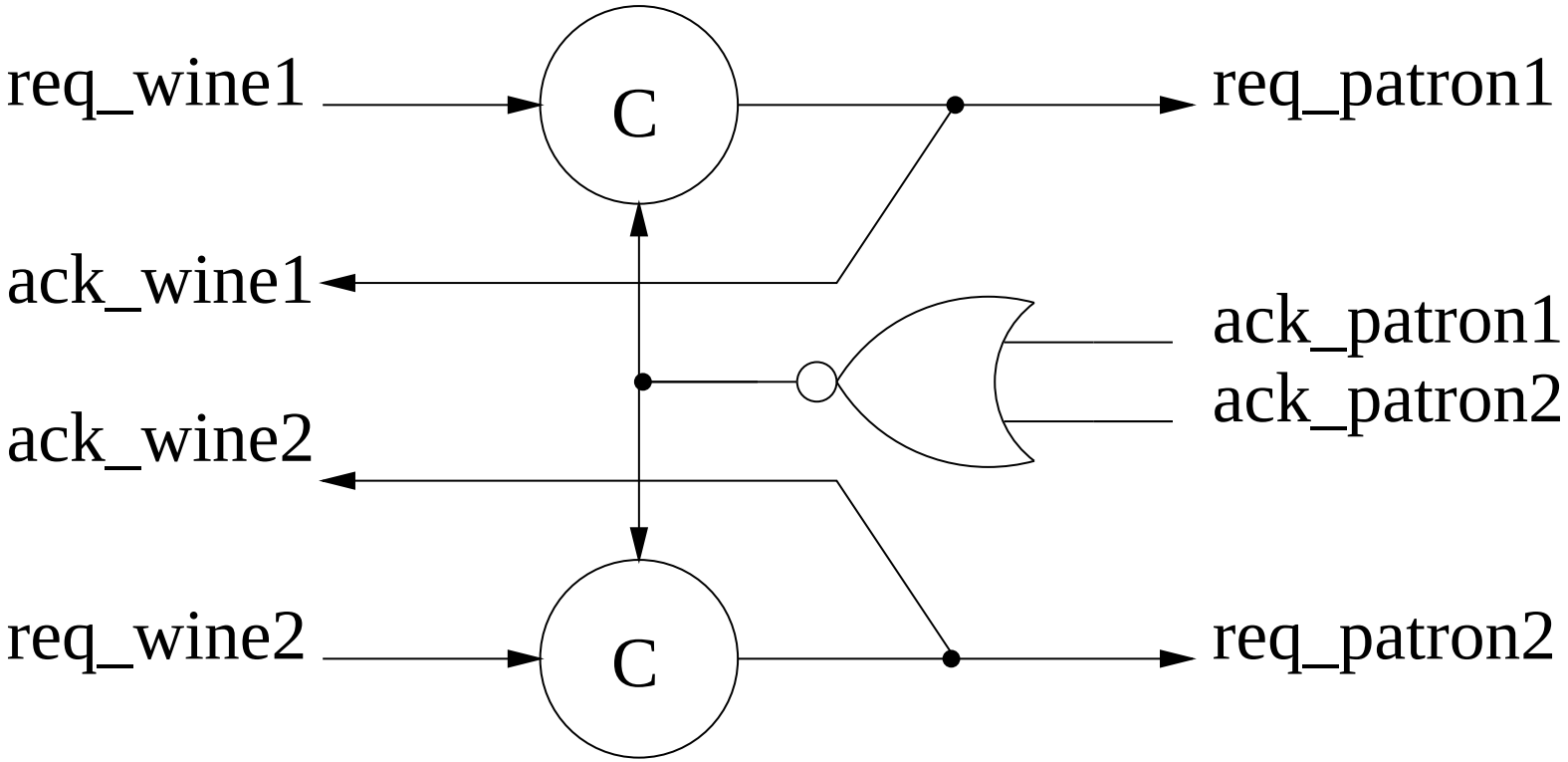
Testing Asynchronous Circuits (Good News)

- Since many asynchronous styles use handshakes, for many faults, a defective circuit simply halts.
- In the *stuck-at fault model*, a defect is assumed to cause a wire to become permanently *stuck-at-0* or *stuck-at-1*.
- If acknowledge wire is stuck-at-0, the request is never acknowledged, causing the circuit to stop and wait forever.
- Easy to detect with a timeout.
- The circuit is said to be *self-checking*.
- Delay-insensitive circuits halt in the presence of any stuck-at fault on any wire in the design.
- Muller circuits halt for any *output stuck-at fault*.

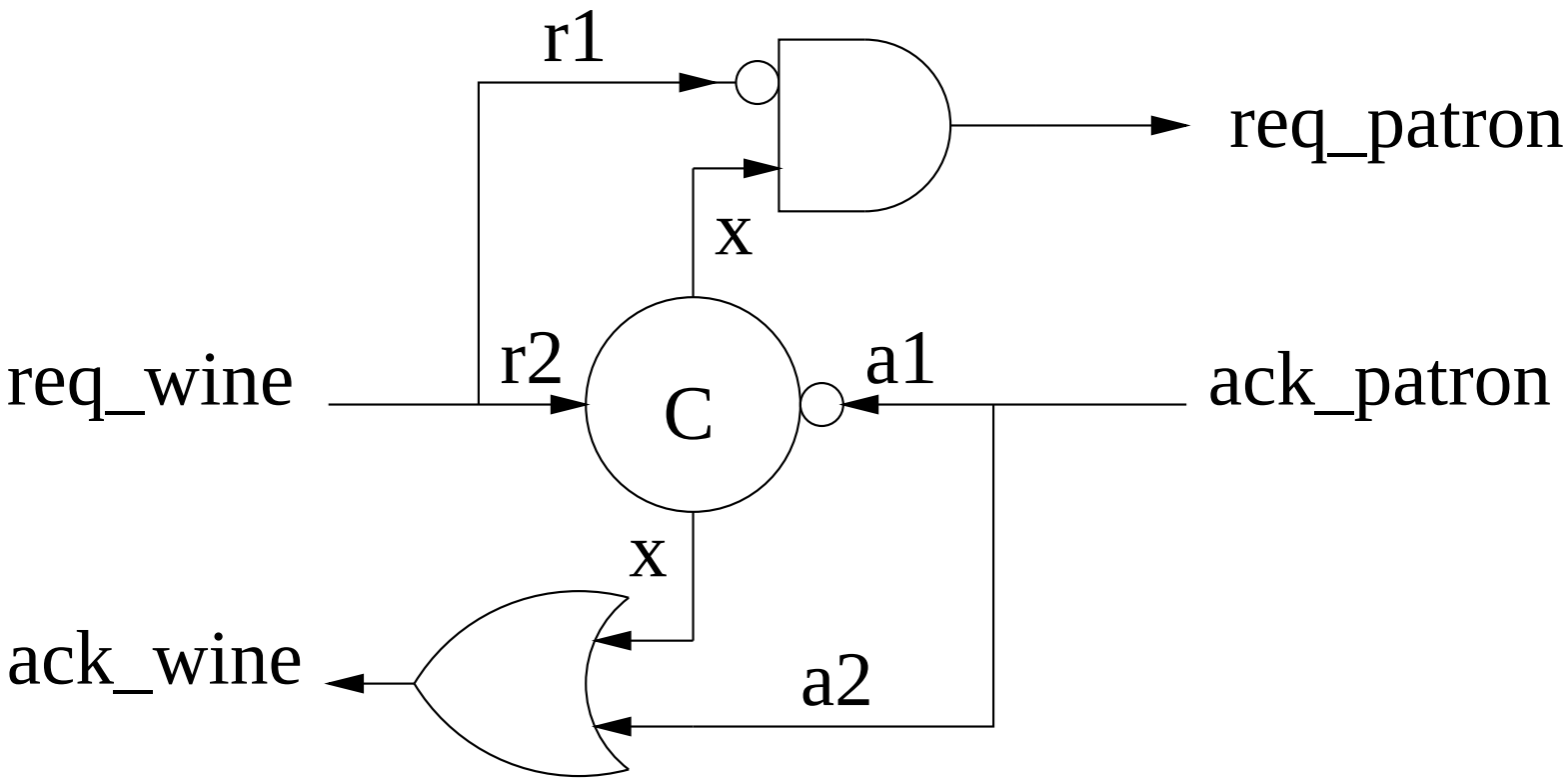
Fault Locations



Testing Example 1



Testing Example 2



Consider $r1$ stuck-at-0

$req\_wine+, x+, req\_patron+$

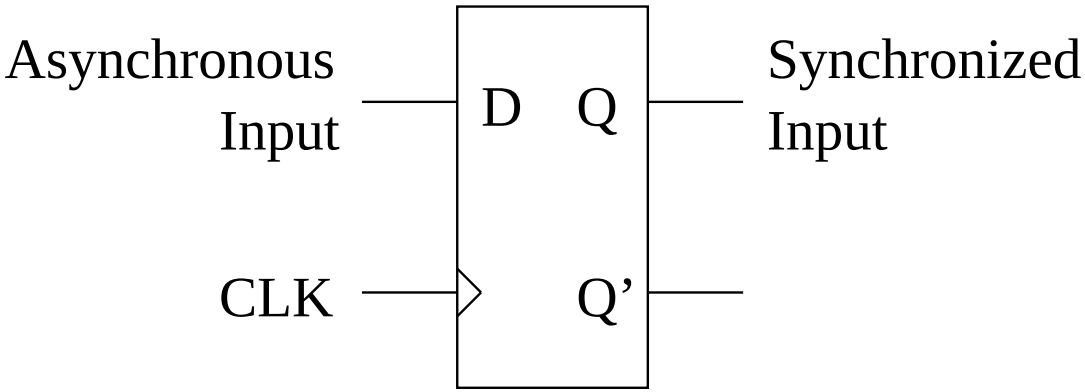
Other Testing Issues

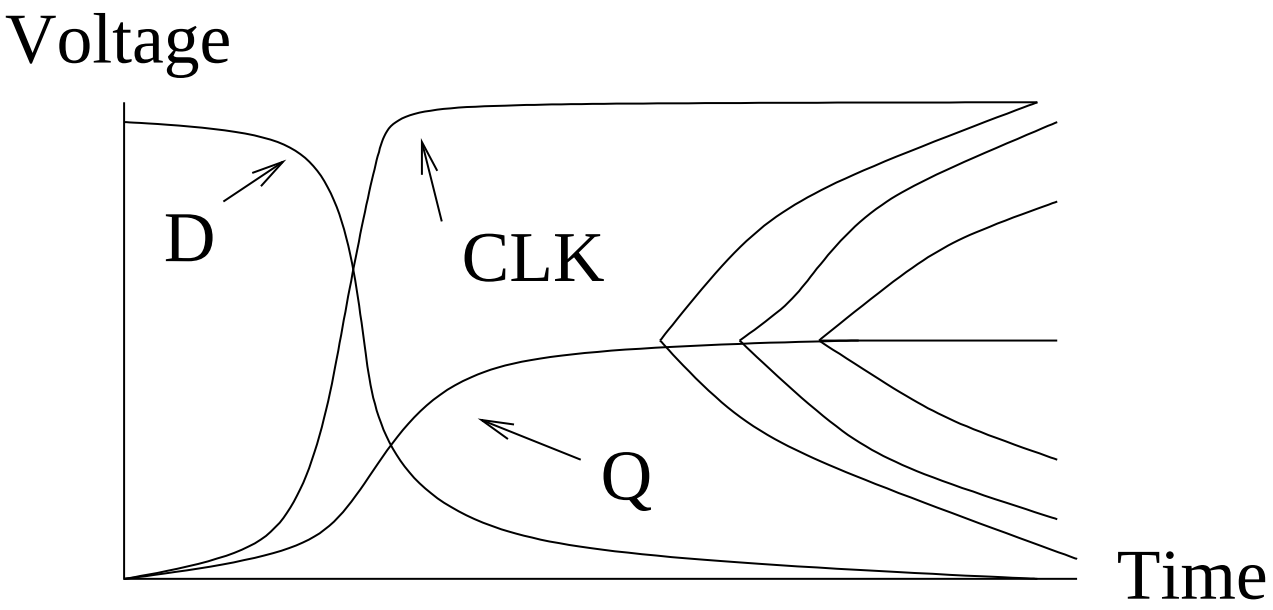
- In the *isochronic fork fault model*, faults can be detected on all branches of a nonisochronic fault by the circuit halting, but only on the input to forks that are isochronic.
- More complex *delay fault* and *bridging fault* models have been successfully adapted to asynchronous circuits.
- Once one selects a fault model, next step is to add circuitry to apply and analyze test vectors, such as *scan paths*.
- Must also generate sufficient set of test vectors to guarantee a high degree of coverage of all possible faults in our model.
- While traditional synchronous test methods don't work off the shelf, many popular methods adapted to asynchronous problem.

The Synchronization Problem

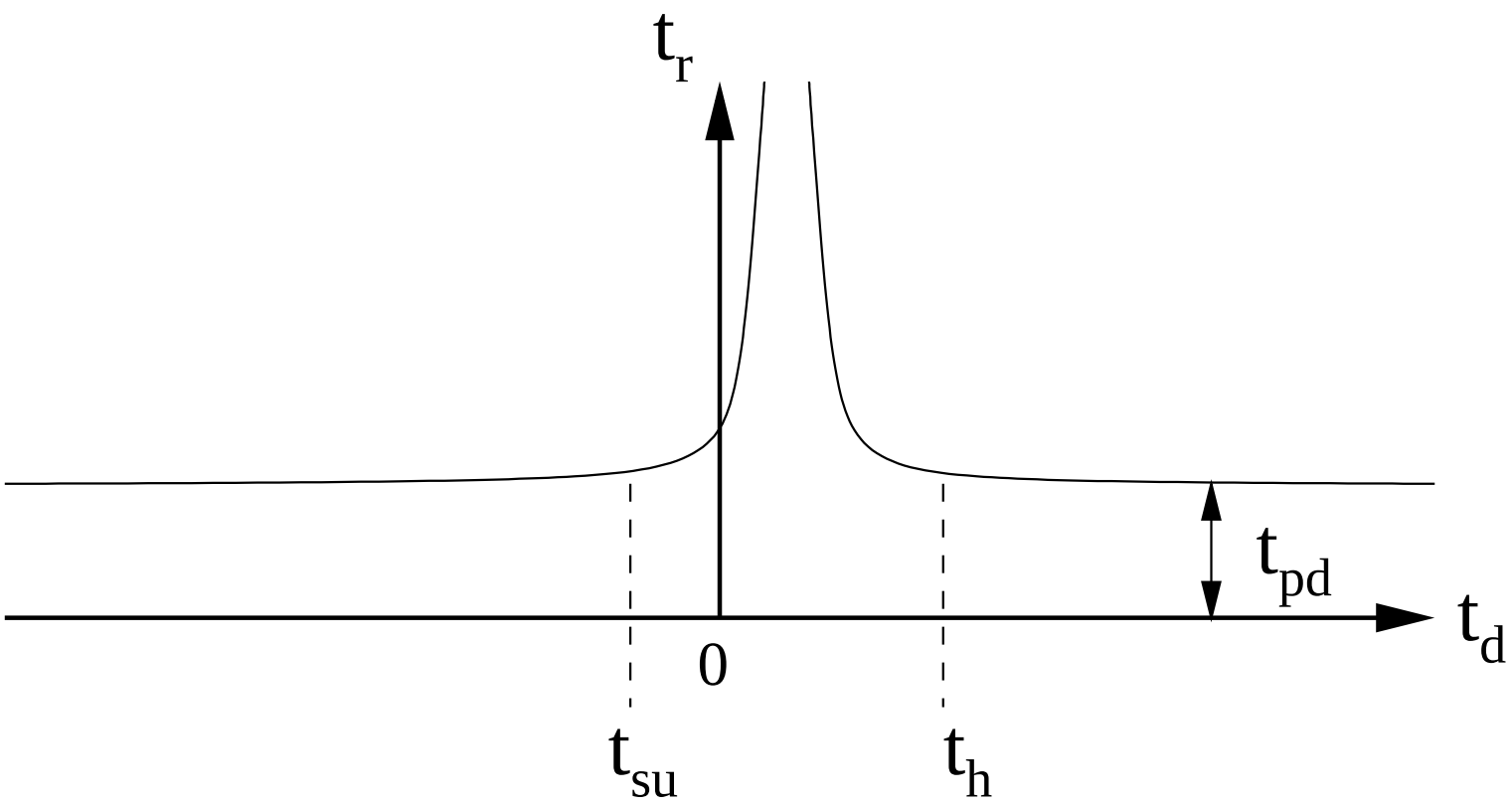
- It is difficult to reliably communicate between asynchronous and synchronous modules without substantial latency penalties.
- Synchronous circuit sampling asynchronous signal changing too close to the clock edge may end up in a *metastable state*.
- If this state persists so long that something bad happens, this is called a *synchronization failure*.

Metastability





Flip-Flop Response Time



Probability of Synchronization Failure

- Data arrive at time uniformly distributed within clock cycle, T.

$$P(t_d \in [t_{su}, t_h]) = \frac{t_h - t_{su}}{T} \quad (1)$$

- Assume flip-flop is given bounded amount of time, t_b:

$$P(t_r > t_b \mid t_d \in [t_{su}, t_h]) = \frac{1}{k + (1-k)e^{(t_b - t_{pd})/\tau}} \quad (2)$$

- k is a positive fraction less than 1 and τ is a time constant with values on the order of a few picoseconds for modern technologies.
- Combine Equations 1 and 2 using Bayes rule:

$$P(t_r > t_b) = P(t_d \in [t_{su}, t_h]) \cdot P(t_r > t_b \mid t_d \in [t_{su}, t_h]) \quad (3)$$

$$= \frac{t_h - t_{su}}{T} \cdot \frac{1}{k + (1-k)e^{(t_b - t_{pd})/\tau}} \quad (4)$$

Probability of Synchronization Failure (cont)

- If $t_b - t_{pd} \geq 5\tau$, Equation 4 can be simplified as follows:

$$P(t_r > t_b) \approx \frac{t_h - t_{su}}{T} \cdot \frac{e^{-(t_b - t_{pd})/\tau}}{1-k} \quad (5)$$

- By combining constants, Equation 5 can be changed to

$$P(t_r > t_b) \approx \frac{T_0}{T} \cdot e^{-t_b/\tau} \quad (6)$$

- T_0 and τ scale linearly with feature size.
- Equation 6 has been verified experimentally and found to be a good estimate as long as t_b is not too close to t_{pd}.
- There is no finite value of t_b such that $P(t_r > t_b) = 0$.

Synchronization Error

- A *synchronization error* occurs when t_r is greater than time available to respond, t_a.
- A synchronization failure occurs when there is an inconsistency caused by the error.
- Expected number of errors is

$$E_e(t_a) \quad = \quad P(t_r > t_a) \cdot \lambda \cdot t \tag{7}$$

where λ is the average rate of change of the signal being sampled and t is the time over which the errors are counted.

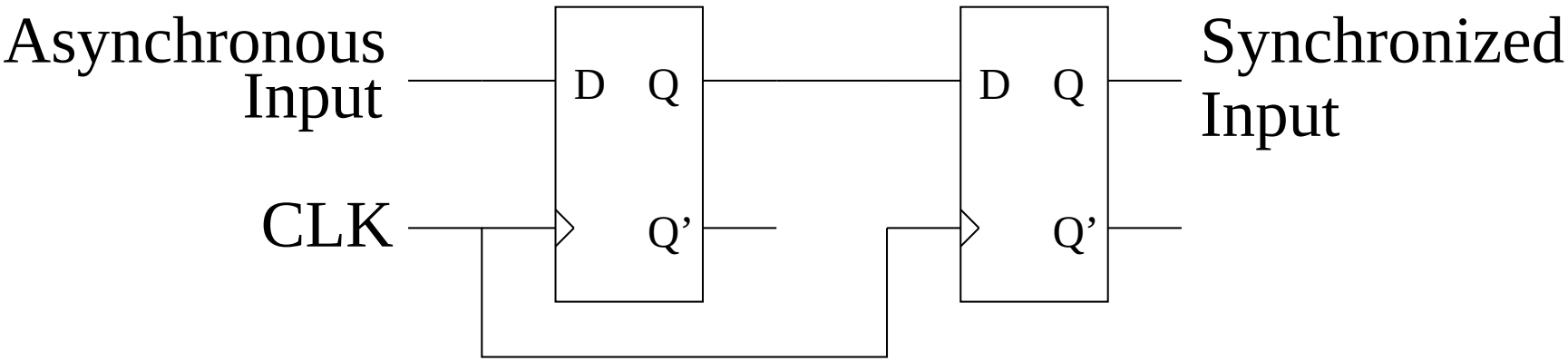
Mean Time Between Failure

- If we set $E_e(t_a)$ to 1, change t to MTBF (*mean time between failure*), substitute Equation 6 for $P(t_r > t_a)$, and rearrange Equation 7, we get

$$\text{MTBF} = \frac{T \cdot e^{t_a/\tau}}{T_0 \cdot \lambda} \tag{8}$$

- This equation increases rapidly as t_a is increased.
- Though no absolute bound in which no failure can ever occur, there does exist an engineering bound.

Double Latch Solution



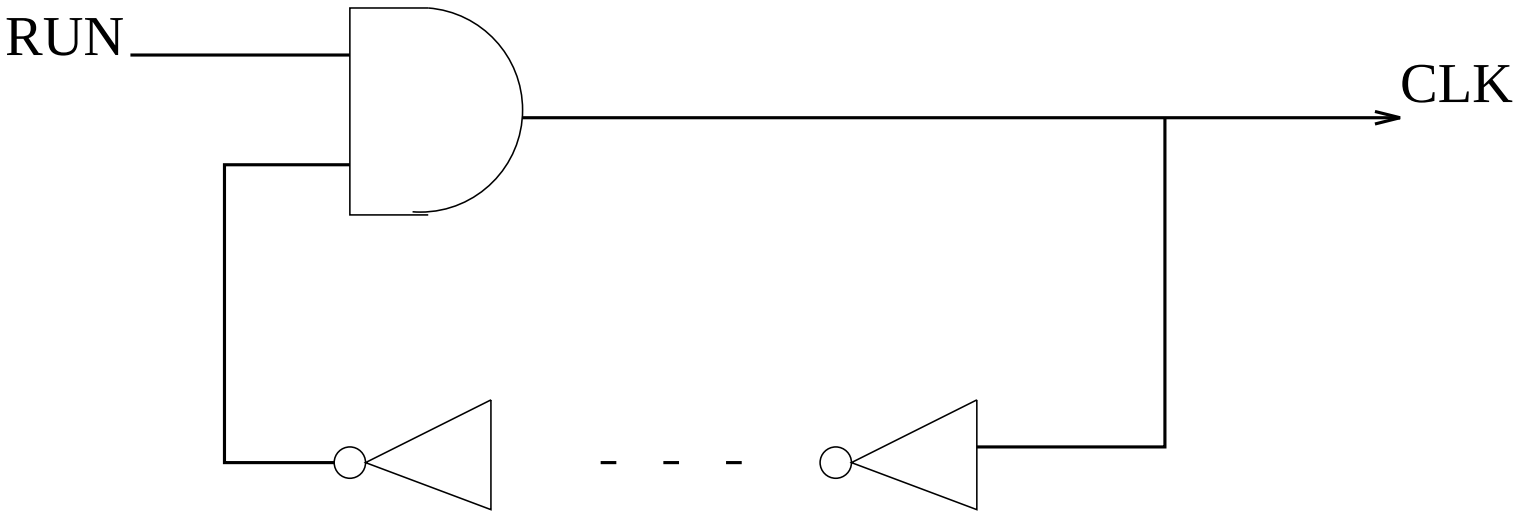
Reducing the Probability of Failure

- If n extra latches are added in series with an asynchronous input, the new value of t_a is given by

$$t_a' = t_a + n(T - t_{pd}) \tag{9}$$

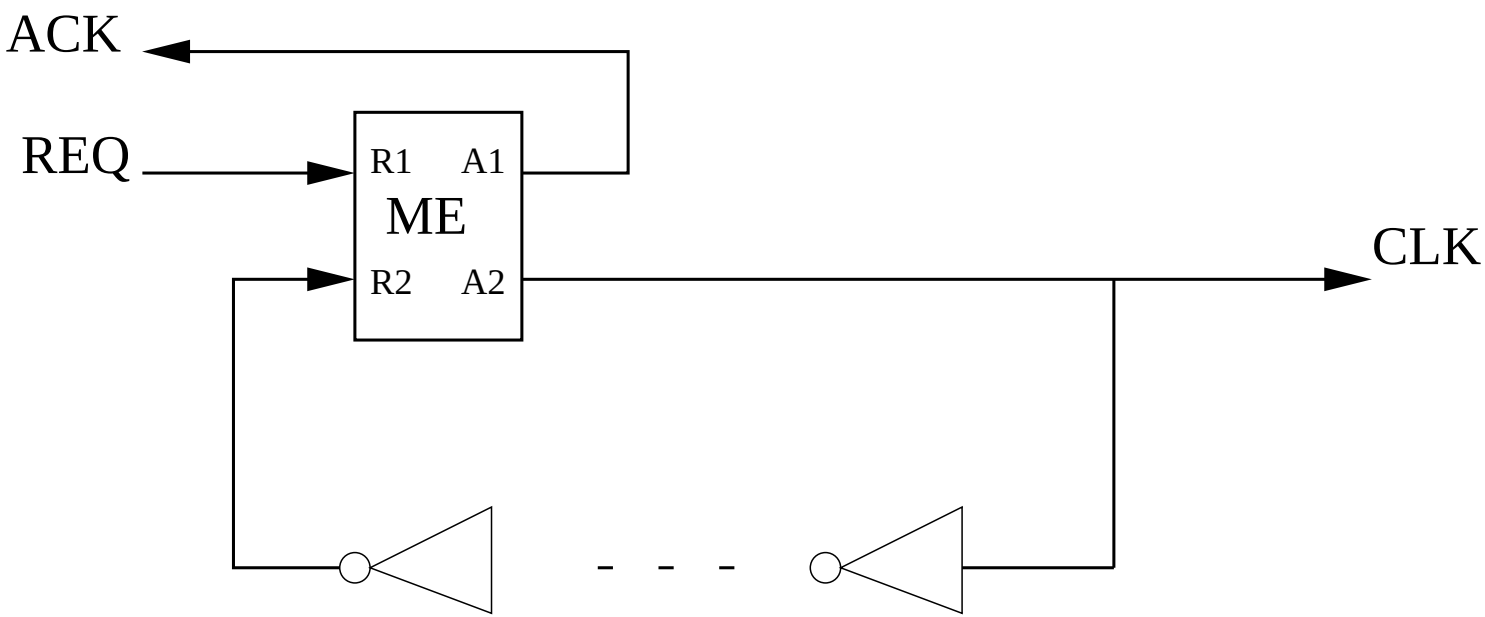
  where T is the clock period and t_{pd} is the propagation delay through the added flip-flops.
- IMPORTANT: INCORRECT IN THE BOOK.
- Cost is extra n cycles of delay when communicating data from an asynchronous module to a synchronous module.
- This scheme only minimizes probability and does not eliminate the possibility of synchronization failure.

Stoppable Ring Oscillator Clock



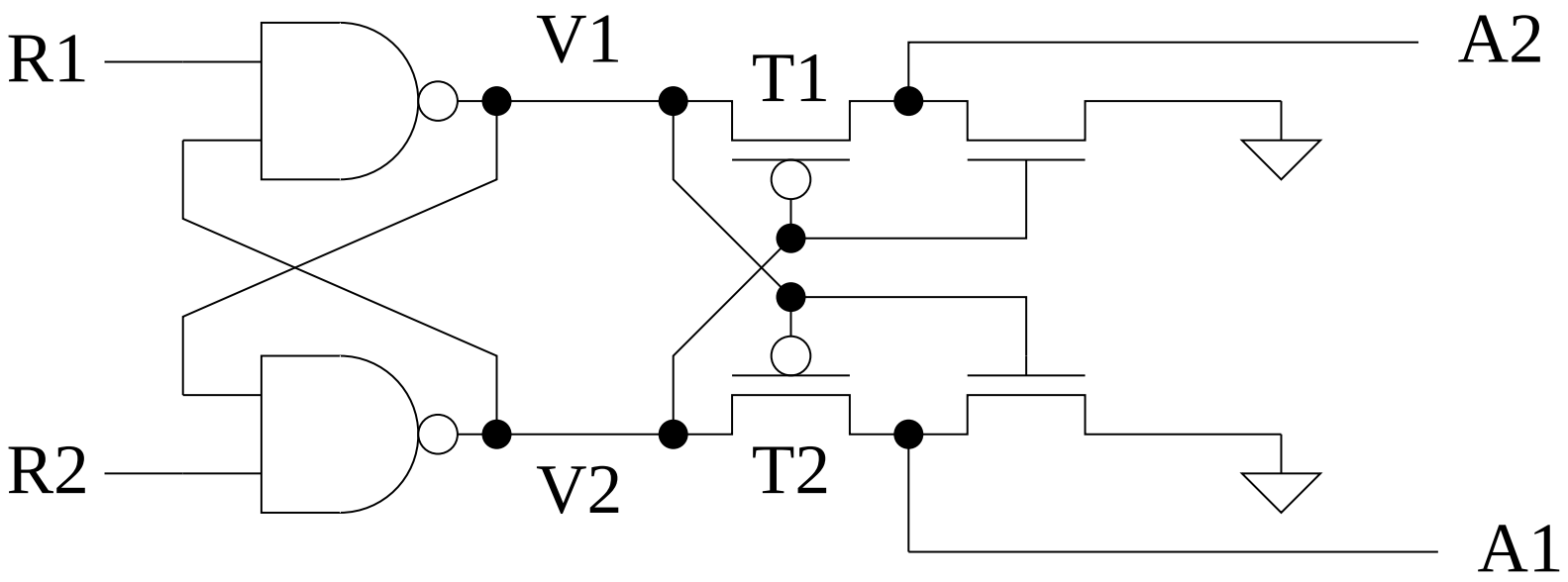
Odd number of inverters

Stoppable Ring Oscillator Clock with ME

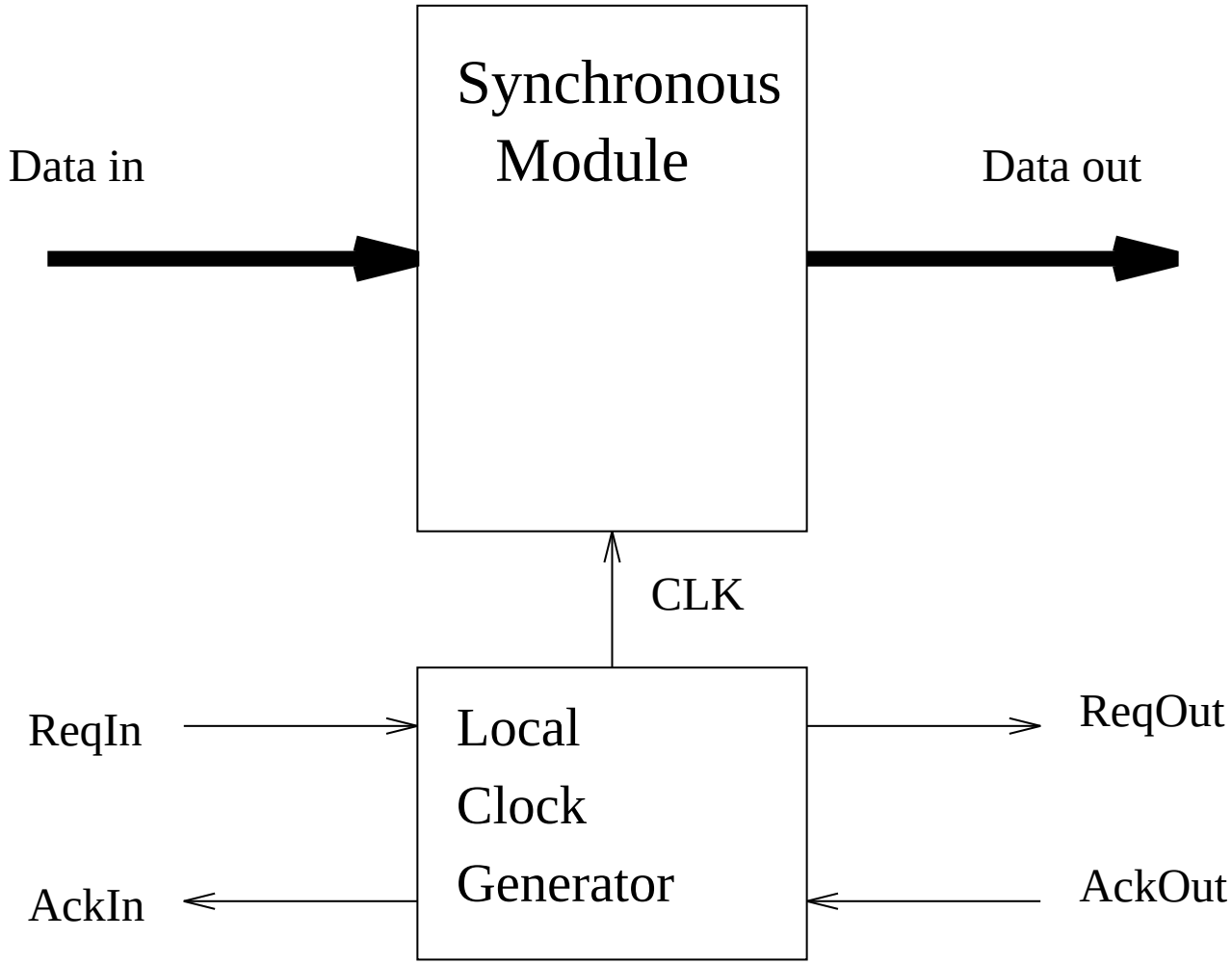


Odd number of inverters

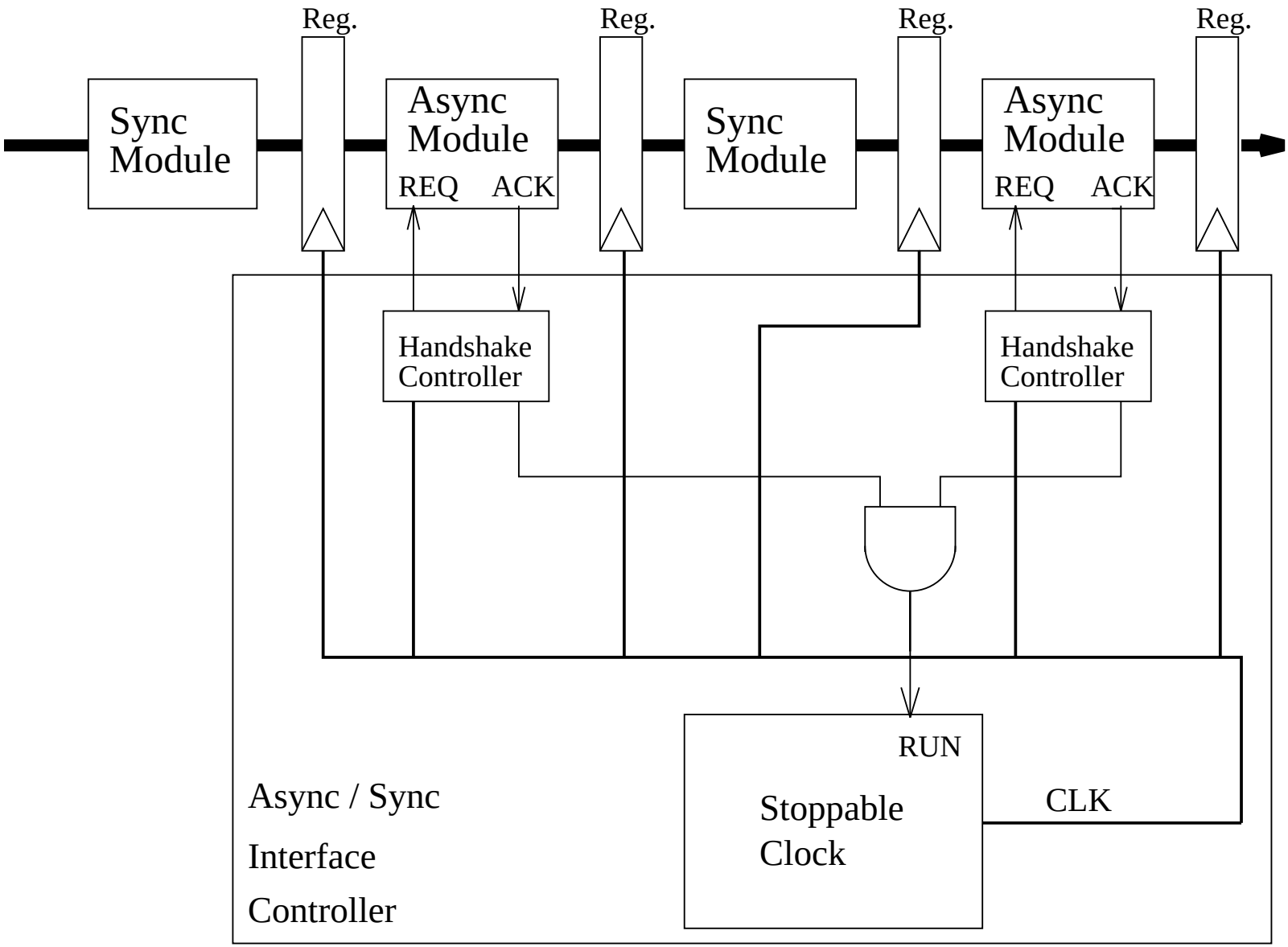
Circuit for Mutual Exclusion



Basic Module of a GALS Architecture



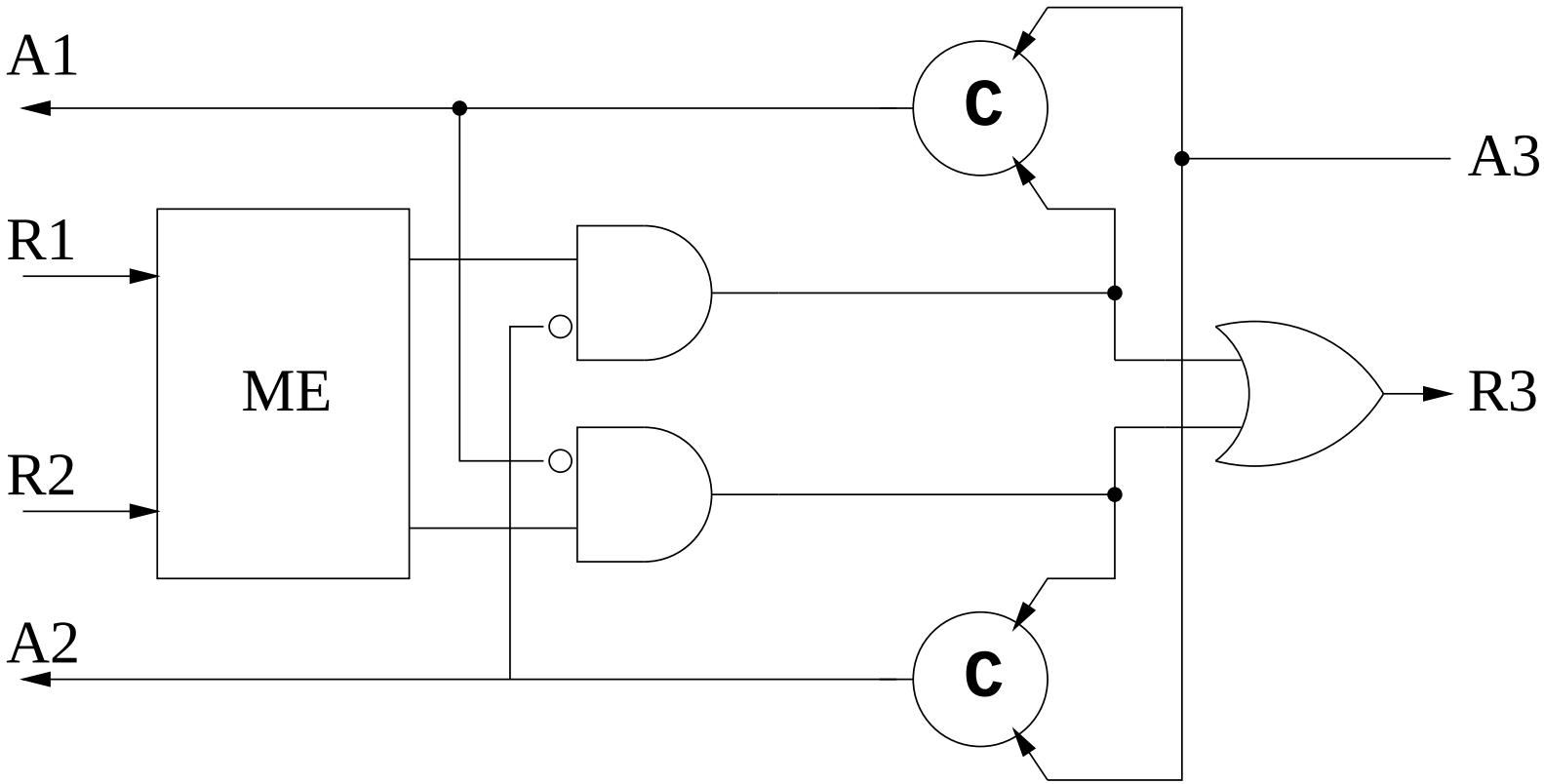
LAGS Architecture



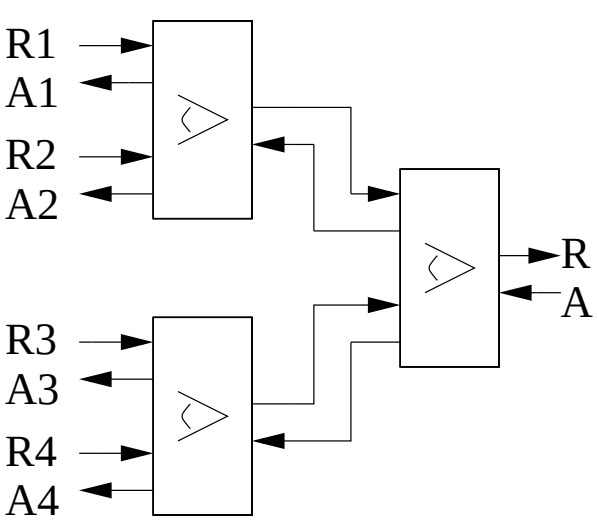
Arbitration

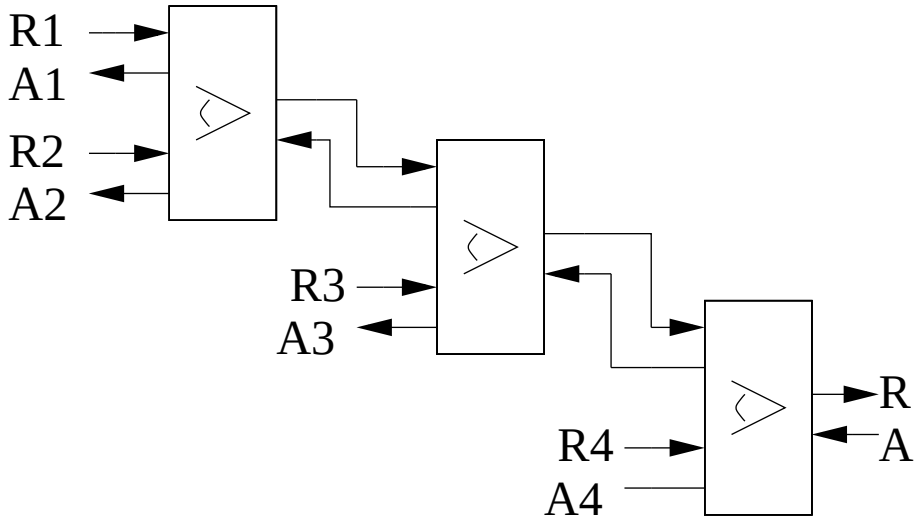
- *Arbitration* necessary when two or more modules want mutually exclusive access to a shared resource.
- If modules are asynchronous, arbitration also cannot be guaranteed in a bounded amount of time.
- If all modules are asynchronous, it is possible to build an *arbiter* with zero probability of failure.

Circuit for Arbitration



Four-Way Arbiters





Future of Asynchronous Design

- Papers in 60s and 70s cite same advantages of asynchronous design used in papers today.
- Synchronous design has been so simple to understand and use.
- Asynchronous revolution may be quite gradual.
- Even if it never happens asynchronous research has been useful.
- Hopefully when you reach industry, you will consider using asynchronous design!