EECS 373

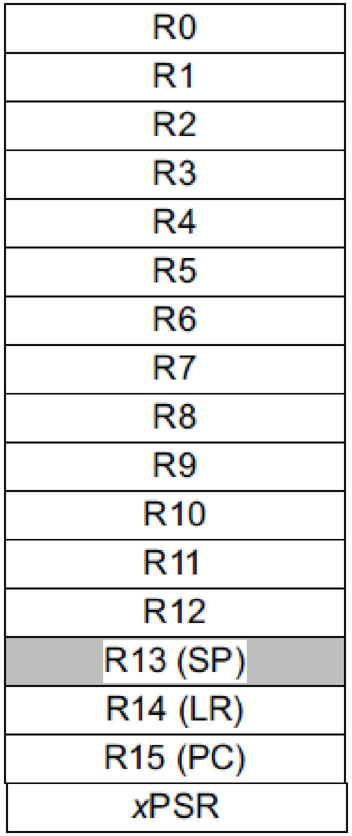
Introduction to Embedded System Design

Robert Dick

University of Michigan

Lecture 4: Debugging complex systems, APB

25, 30 January 2024



R0
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R12
R13 (SP)
R14 (LR)
R15 (PC)
*x*PSR

Outline

- **Project idea areas**
- Memory-Mapped I/O Review
- Compile-time error checking example
- Debugging Complex Systems
- Advanced Peripheral Bus

General project idea areas

- * Biological monitoring and control
- Cleaning
- Emergency response
- Fashion
- Music
- * Personal monitors and assistants
- * Smart home and cooking
- Sports and games
- Translators
- Transportation
- UI

Lecture tuning

- Pace: 4 say slow down, 10 say keep as-is, 2 say speed up.
  - Resolution: slow down very slightly.
- Examples: 11 say use fewer examples, 5 say keep as-is, 3 say more examples.
  - Resolution: Reduce % of time spent on examples and avoid marathon sessions, but don’t stop completely.
- Slides and DocCam: 8 say clear, 5 say unclear.
  - Resolution: Check DocCam setup carefully.
- Can hear professor: 13 say yes, 0 say no.
- Can hear students: 4 say yes, 7 say no.
  - Resolution: Repeat student questions. Also, please ask loudly.
- Synchronization: 7 say synchronized, 5 say not synchronized.
  - Will discuss synchronization in weekly staff meetings but not sure what to do about this one. Can’t do perfectly with lab times spanning a week.
- Professor cares about teaching me to be a better computer engineer: 16 yes, 1 no.
  - If it doesn’t seem to be the case, please come to office hours. I can help on most computer engineering related topics.

Outline

- Project idea areas
- **Memory-Mapped I/O Review**
- Compile-time error checking example
- Debugging Complex Systems
- Advanced Peripheral Bus

Example

```
#include <stdio.h>
#include <inttypes.h>

#define REG_FOO 0x40000140

int main(void) {
  volatile uint32_t *reg = (uint32_t *)(REG_FOO);
  *reg += 3;

  print_uint(*reg);
  return 0;
}
```

“*reg += 3” is turned into a ld, add, str sequence

- Load instruction.
  - A bus read operation commences.
  - The CPU drives the address “reg” onto the address bus.
  - The CPU indicated a read operation is in process (e.g., R/W#).
  - Some “handshaking” occurs.
  - The target drives the contents of “reg” onto the data lines.
  - The contents of “reg” are loaded into a CPU register (e.g., r0).
- Add instruction.
  - An immediate add (e.g., add r0, #3) adds three to this value.
- Store instruction.
  - A bus write operation commences.
  - The CPU drives the address “reg” onto the address bus.
  - The CPU indicated a write operation is in process (e.g., R/W#).
  - The CPU drives the contents of “r0” onto the data lines.
  - Some “handshaking” occurs.
  - The target stores the data value into address “reg”.

Outline

- Project idea areas
- Memory-Mapped I/O Review
- **Compile-time error checking example**
- Debugging Complex Systems
- Advanced Peripheral Bus

Outline

- Project idea areas
- Memory-Mapped I/O Review
- Compile-time error checking example
- **Debugging Complex Systems**
- Advanced Peripheral Bus

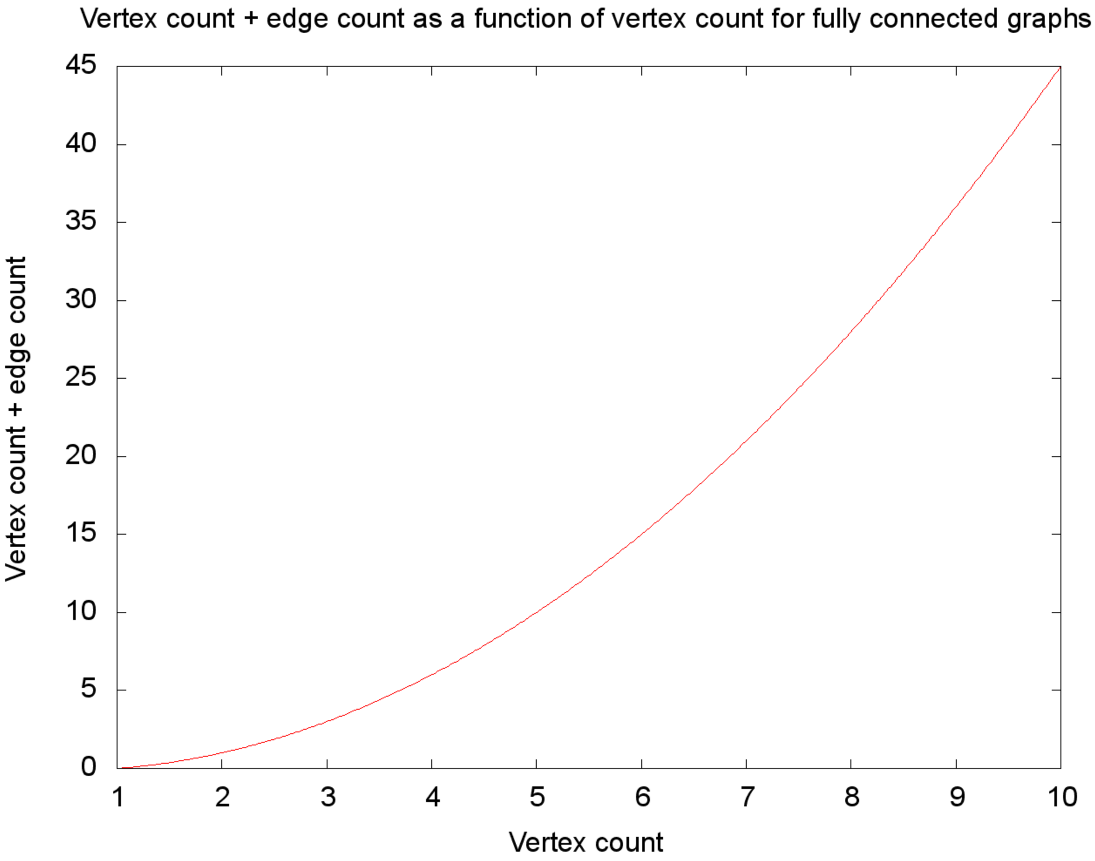
Design and debugging: computer systems are graphs

- This is quick and seems simple but it is actually deep and important.

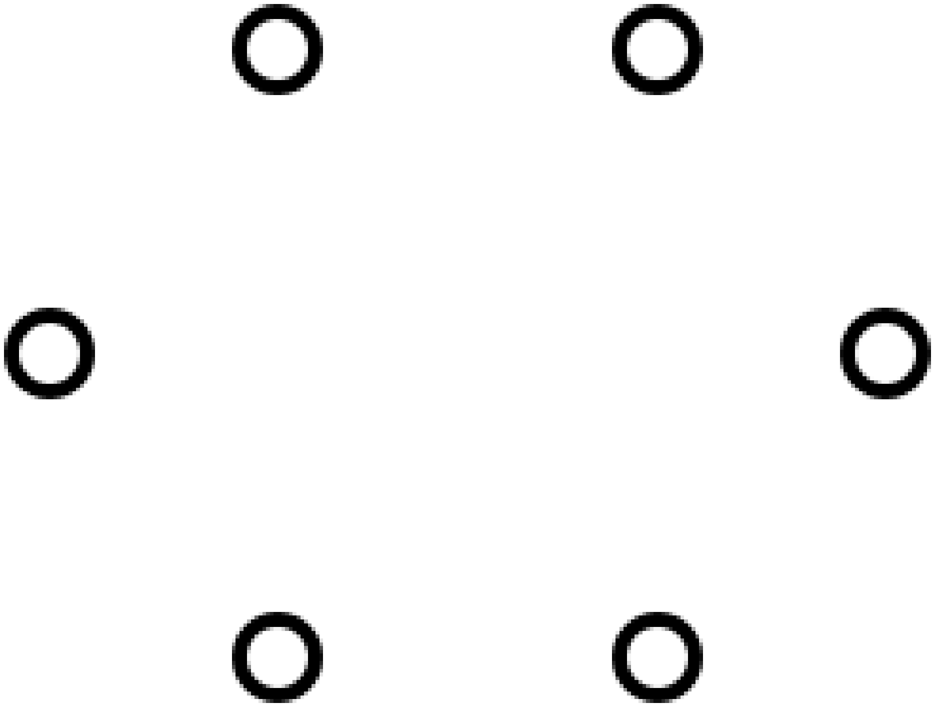
- A computer system is a graph.
- Each component, e.g., a line of code or transistor, is a vertex (v).
- Each effect that influences other components is an edge (e).
- Complexity is a function of |v| + |e|.

Graph sizes

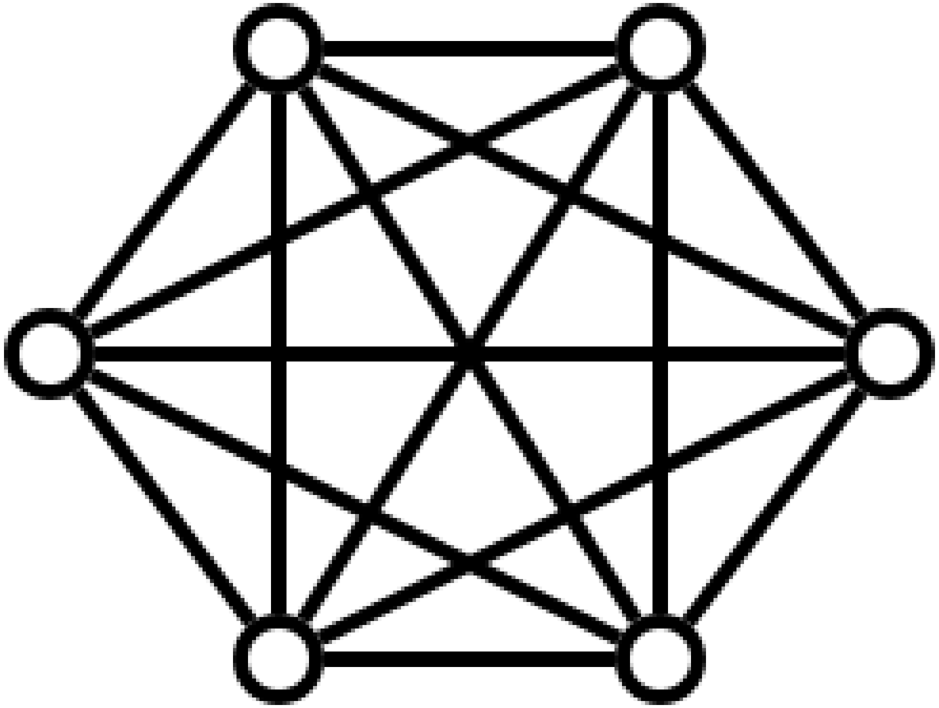
- For undirected fully connected graphs.
  - $|e| = |v|(|v| - 1) / 2$
- But it's much worse than that because your ability to analyze systems decreases dramatically with system size.
- So system complexity (debug time) is a superlinear function of |v|, like $|v|^k$.
- k is generally >= 2, and probably quite a bit bigger.



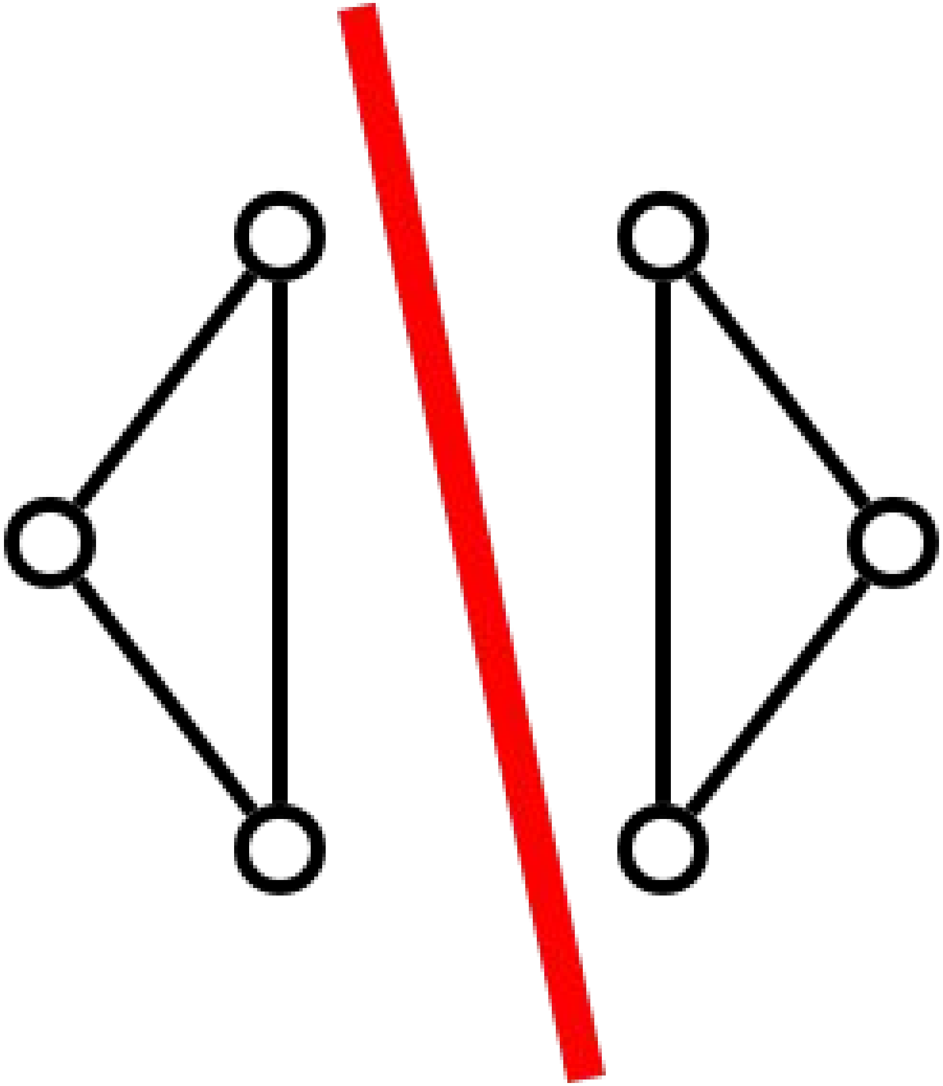
Best-case complexity



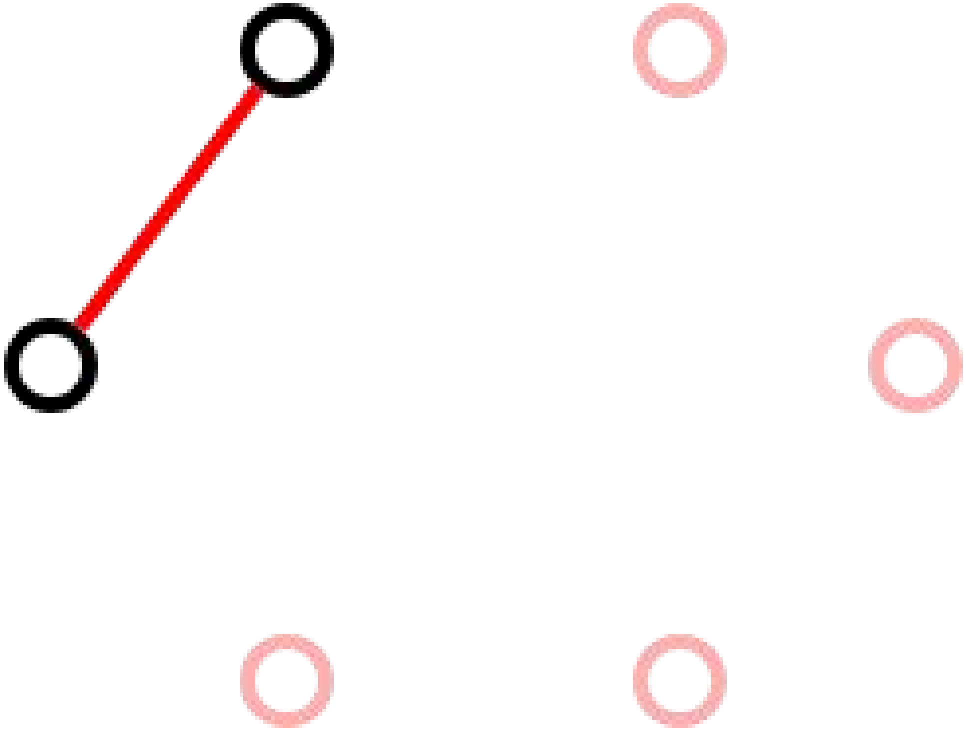
Worst-case complexity



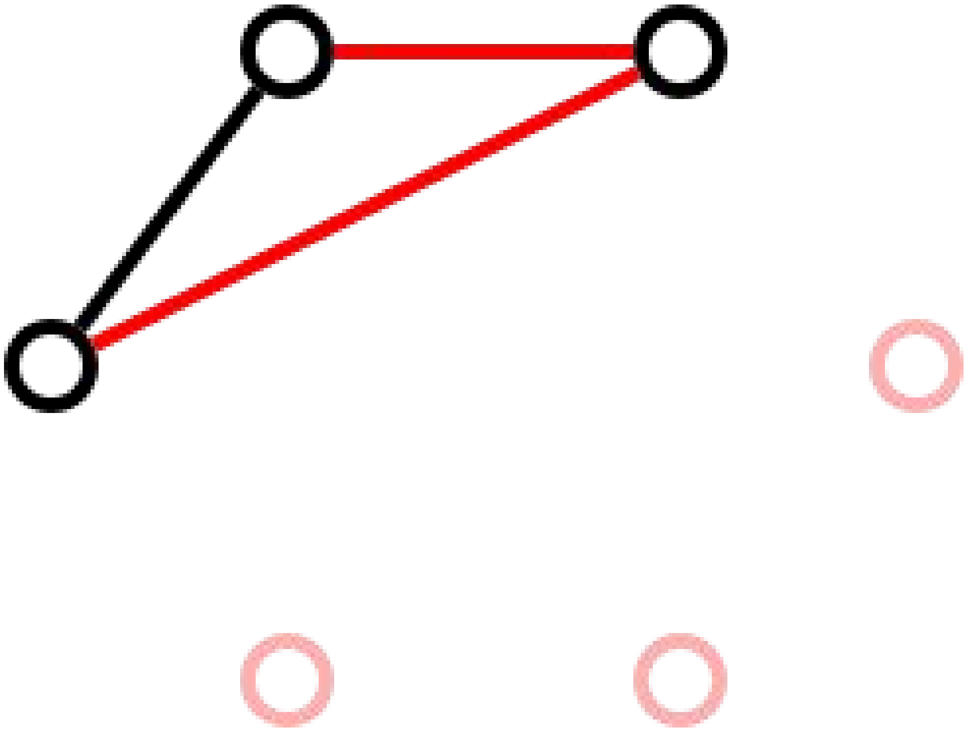
Managing complexity



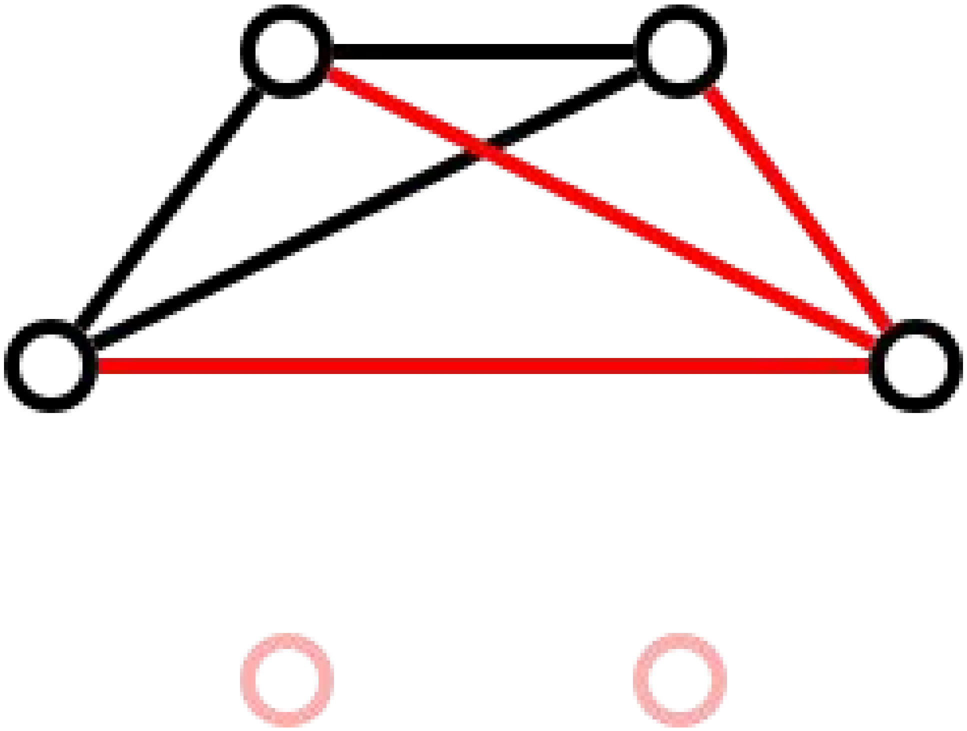
Incremental tested growth



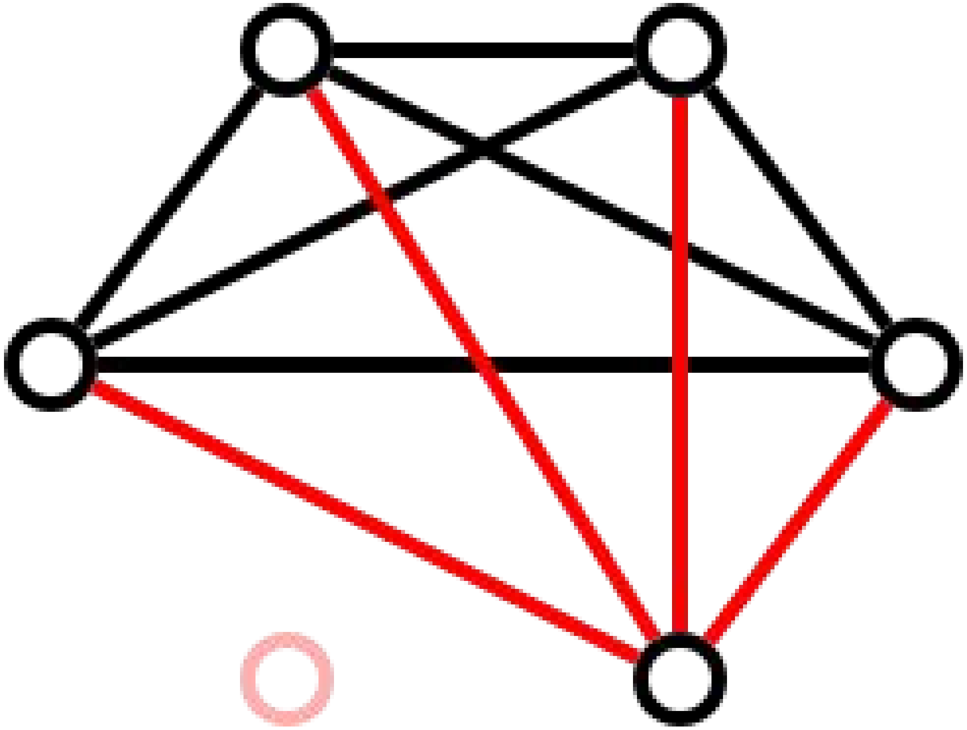
Incremental tested growth



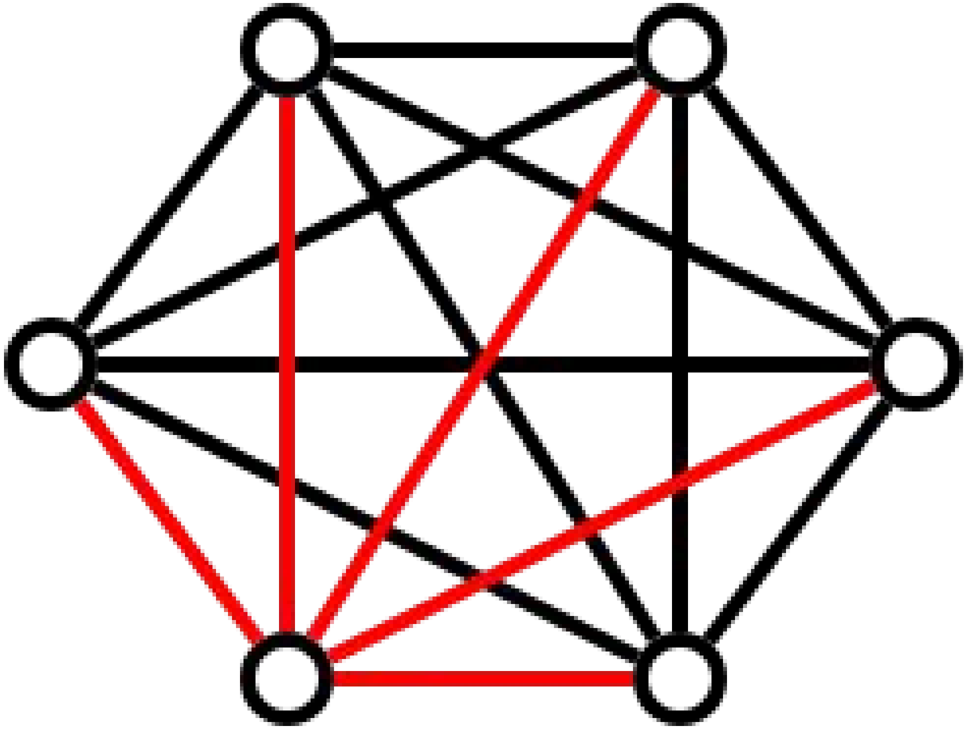
Incremental tested growth



Incremental tested growth



Incremental tested growth



Design and debugging: how to make your life easy and make your embedded systems work

- Control |v|
  - Get a very simple version of the system tested and functioning and add to it in small pieces, testing after each addition.
  - Never build something big and then start testing.
- Control |e|
  - Build and test isolated, side-effect free components with narrow and easy-to-understand interfaces.

Start from the root

- Check the foundation before the roof.
- Check the power distribution network integrity before checking the software.

Switch between information gathering and reasoning

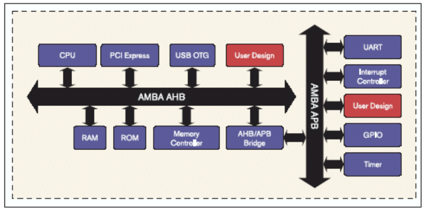
- Search process.
- Large search space.
- Probing specific locations is expensive.
- Initial conditions.
  - Don't have the data necessary to understand the problem.
  - Haven't done the analysis necessary to convert those data to information.
- Don't neglect either weakness. Iterate.
  - Conduct naïve experiments to gather information.
  - Stop testing and reason about problem, using conclusions to devise additional tests.
- Most engineers are better at analysis or testing.
  - Don’t stay under the streetlight.

Outline

- Project idea areas
- Memory-Mapped I/O Review
- Compile-time error checking example
- Debugging Complex Systems
- **Advanced Peripheral Bus**

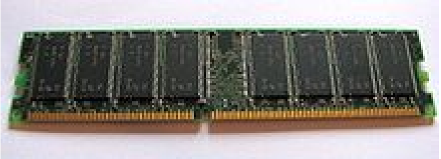
Details of the bus “handshaking” depend on the particular memory/peripherals involved

- SoC memory/peripherals
  - AMBA AHB/APB

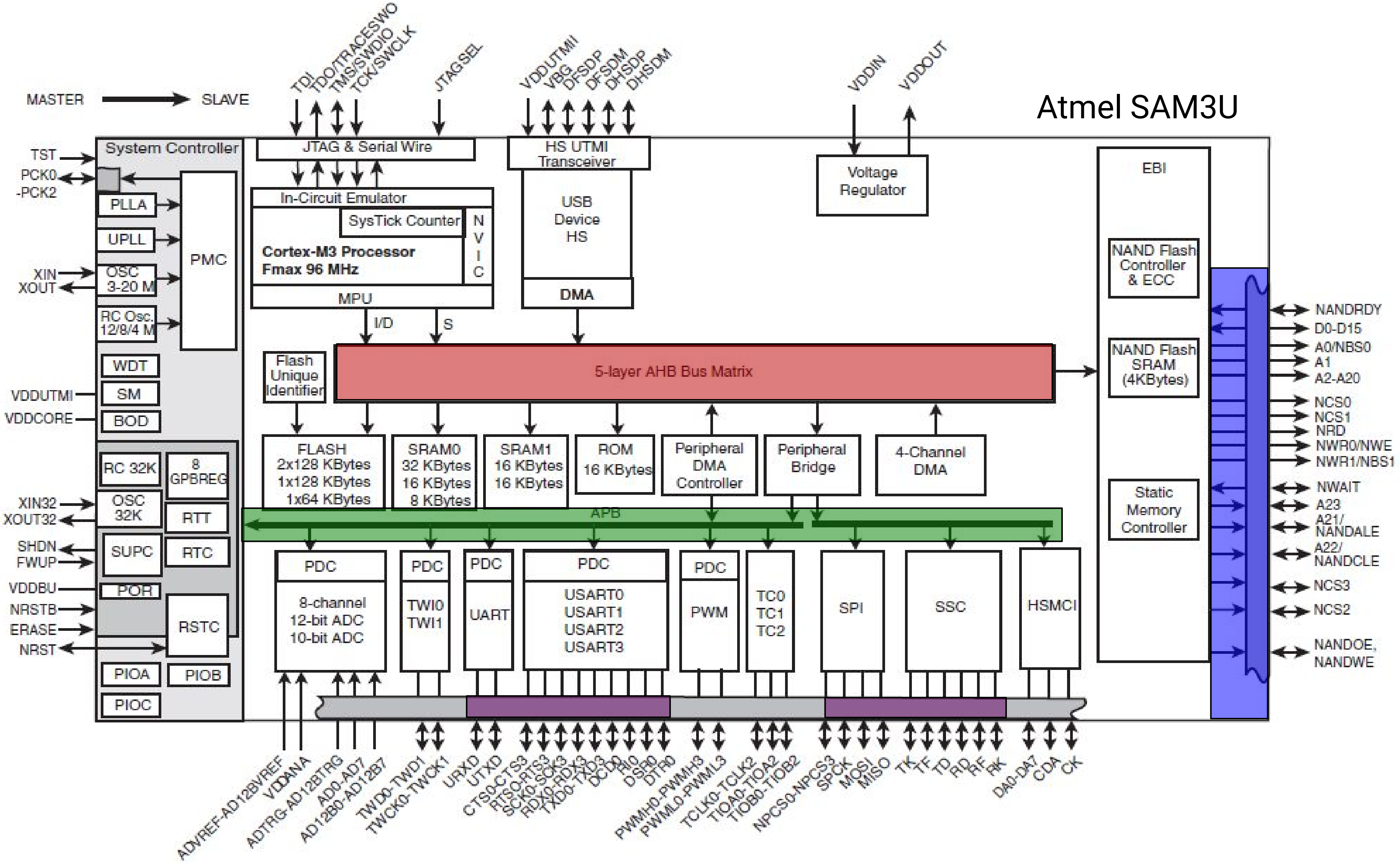


- NAND Flash
  - Open NAND Flash Interface (ONFI)

- DDR SDRAM
  - JEDEC JESD79, JESD79-2F, etc.

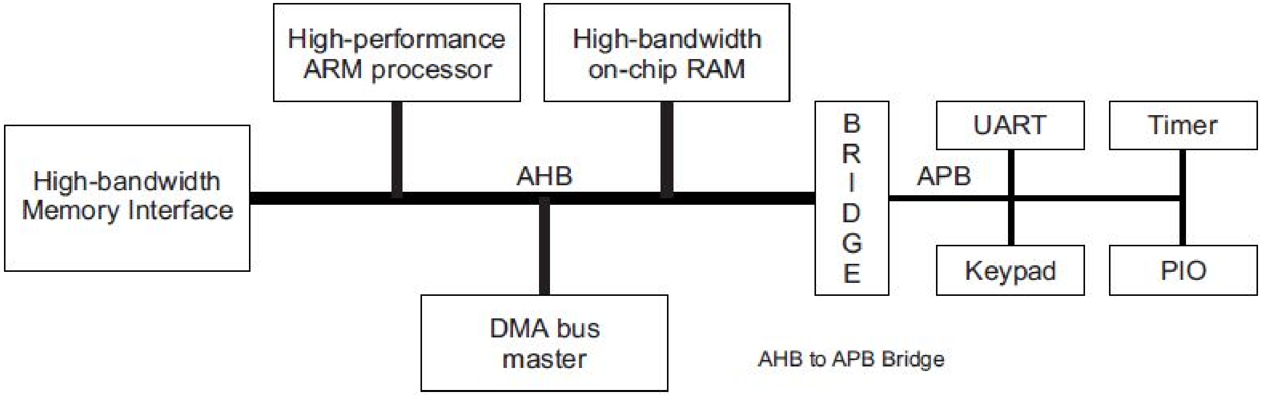


Modern embedded systems have multiple busses



Atmel SAM3U

Advanced Microcontroller Bus Architecture (AMBA)
- Advanced High-performance Bus (AHB)
- Advanced Peripheral Bus (APB)


High-performance ARM processor
High-bandwidth on-chip RAM
High-bandwidth Memory Interface
AHB
B R I D G E
UART
Timer
APB
Keypad
PIO
DMA bus master
AHB to APB Bridge

AHB

- High performance
- Pipelined operation
- Burst transfers
- Multiple bus initiators
- Split transactions

APB

- Low power
- Latched address/control
- Simple interface
- Suitable of many peripherals

STM32 Block Diagram

STM32L4R5xx, STM32L4R7xx and STM32L4R9xx Description

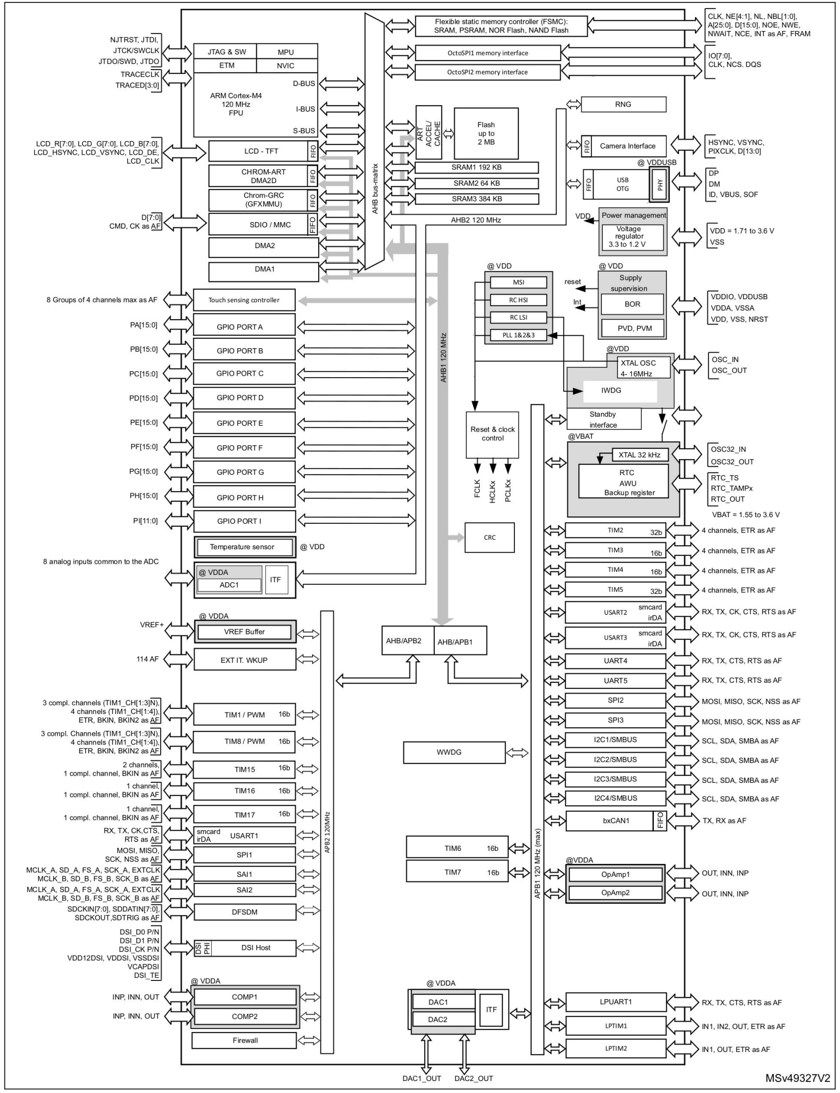


Figure 1. STM32L4R5xx, STM32L4R7xx and STM32L4R9xx block diagram

Note: AF: alternate function on I/O pins.

Bus terminology

Transactions have “*initiators*” and “*targets*”

- Potential initiators, sometimes called “*masters*”.
  - In many cases there is only one bus master (*single master* vs. *multi-master*).
- Non-initiators, sometimes called “slaves”. They can't start transactions, but they carry them out when a master initiates one.
- Some wires might be shared among all devices while others might be point-to-point connections (generally connecting the initiator to each target).

Driving shared wires

- Some shared wires might need to be driven by multiple devices.
- In that case, we need a way to allow one device to control the wires while the others “stay out of the way”.
- Most common solutions are
  - tri-state drivers and
  - open-collector connections.

Another option: avoid shared wires

- Expensive when connecting chips on a PCB as you are paying for pins and wiring area.
- Doable but costs area and time on-chip.

Wire count

- Consider a single-initiator bus with 5 other devices connected and a 32-bit data bus.
- Shared bus → 32 pins
- Separate buses
  - Each target would need ____ pins for data
  - The initiator would need ____ pins for data
- Pins and wiring area cost money.

APB is designed for ease of use

- Low-cost.
- Low-power.
- Low-complexity.
- Low-bandwidth.
- Non-pipelined.
- Ideal for peripherals.

Done.