Embedded Systems

Sivan Toledo
School of Computer Science
Tel-Aviv University

Students' Expectations

- Before I start talking, what do you hope to learn?

What is An Embedded System?

What is An Embedded System?

- A computer system in a product that is **not** a computer; a machine, appliance, satellite, car, etc
- The primary objective is **not** to process or present information
- Sensing, actuation, control
- Fluid boundaries (e.g. phones)
- More modern buzzwords: sensor networks, cyber-physical systems, …

Embedded Systems are Different

Embedded Systems are Different

- Significant per-unit cost (hardware)
  - Optimize to use fewer/cheaper parts
  - Hardware → software
- Limited power & cost → limited resources
  - Memory, CPU power, communications
- Real time constraints; can satisfy ms or µs constraints
- Hardware/software co-development
  - Also, special hardware for debugging/testing
- Hardware fails
- Unattended operation & limited or no UI

Learning Objectives of the Course

- Skills to design & implement software for an embedded system
- Ability to understand the documentation and to communicate with hardware designers
- Understanding of the deeper issues
  - Concurrency, failures, timing, power, and more
- Not a formal objective but easy to achieve after the course: build simple systems from scratch

More Objectives

- Life at the extreme bottom of the abstraction hierarchy
  - Emphasis on low-level drivers
- Life in a small community of developers
  - Few examples, nobody with your hardware
  - (MCU user groups & vendor support are vibrant)
  - But your skills are more valuable
- Life with a different type of documentation
  - Unfamiliar jargon (as in all tech docs)
  - More complete than software docs

One More

“I Can Build It”

The Structure of the Course

- MSc course & BSc workshop together
- Lectures Sunday 1-4pm, every week
- Labs Sunday 4-6 (maybe 7), every week
- Structured labs in first 7 weeks, project in last 6 weeks (or optional structured labs)
- Projects in pairs (triples only in special cases)
- We negotiate project proposals
- **Hard project deadline: April 15**

Grades & Projects

- You must attend labs & submit lab reports
- Each project is different (and hopefully different from previous years')
  - Creative software products within the boundaries of our hardware (or a bit beyond)
- Grades based on project (penalty for missing labs)
  - Everybody can do well
  - Grade depends on scope & features, difficulty, web presentation, **and oral presentation**

Our Hardware 1: LPC2148 Education Board

- Microcontroller: LPC2148 by NXP
  - ARM7 core, 60MHz, 32-bit, 512KB flash, 40KB RAM, USB device
- Character LCD, serial→USB, SD, joystick & other switches, pots, LEDs, stepper, temp sensor, buzzer
- We have ethernet cards, prototyping cards
- Programming through serial→USB
- We do not use an operating system
- We do not have hardware debugger

Our Hardware 2: Telos B (T-mote sky)

- Microcontroller: MSP430F1611 by TI
  - 8MHz, 16-bit, 48KB Flash, 10KB RAM, low-power
- Serial→USB bridge (also for programming)
- 802.15.4 radio transceiver (low-power data)
- Light, temperature & humidity sensors
- Used in sensor-networking research
- "OS": Contiki (also TinyOS, or no OS)
- No hardware debugger

More Hardware We Have

- Platforms: AVR Butterfly, other LPC boards (CAN), more MSP430 boards, PIC boards
- Some with debuggers
- More low-power RF transceivers (cc1101)
- Wifi boards for embedded systems (designed for PICs, probably usable with other MCUs)
- Android phones coming in (more later)
- Electronic construction & test equipment

Backend Software (& Android)

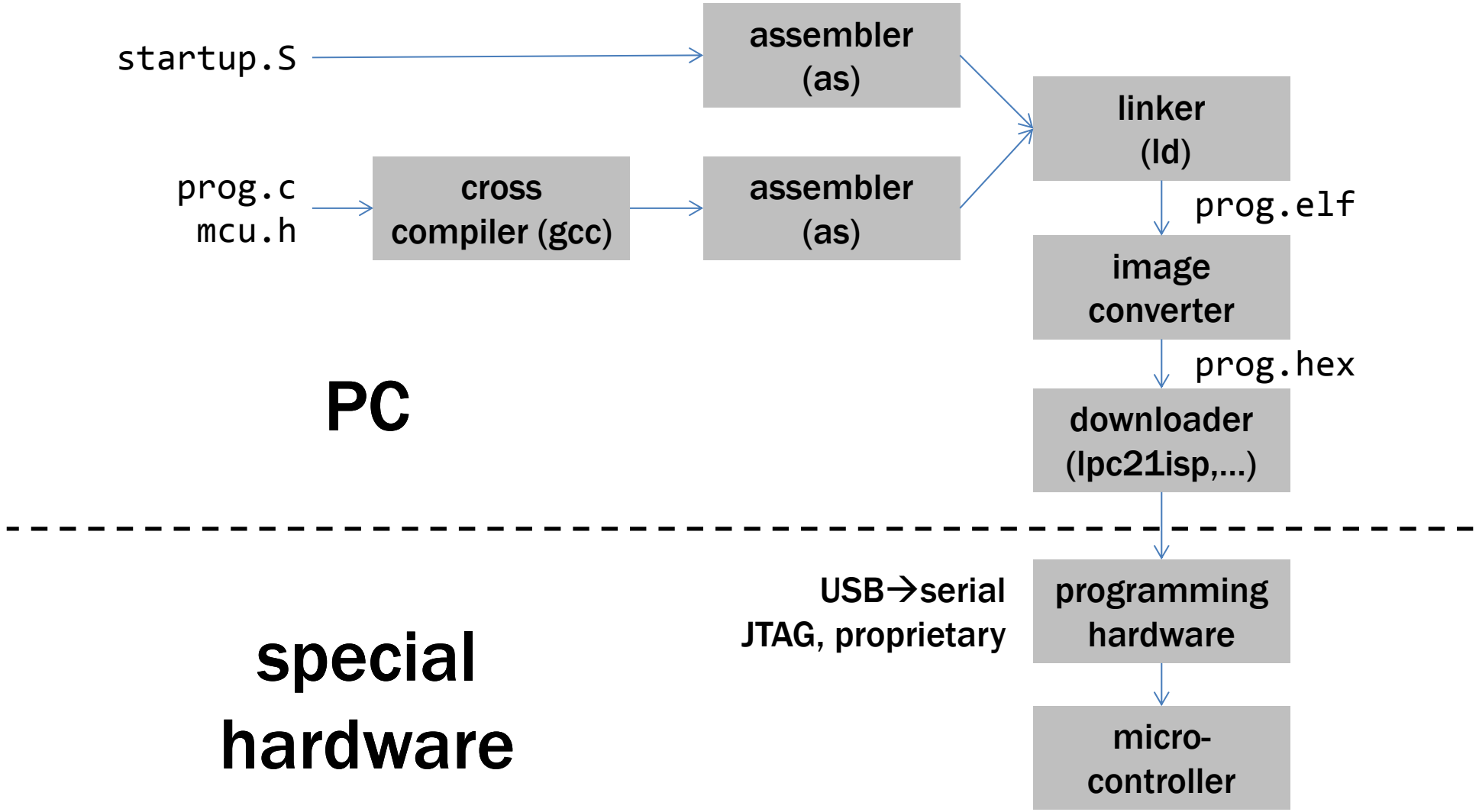
- Some projects are embedded software only
- Some need backend software
- In previous years backend software was all Linux and/or Windows
- This year we can do projects whose backend software runs on an android phone!

2009/2010 Projects

- USB Ethernet Adapter for Windows (RNDIS)
- USB HID host-side Perl libraries
- Live Storage Device
- LUA programming language on the LPC2148
- IR Remote Proxy using a PIC18F2550
- Porting Contiki to the LPC2148 Education Board
- Network-attached storage
- Home automation using XBee modules
- TFTP server over USB (uIP over CDC-ECM)
- Virtual USB Ethernet adapter for LPC2148 (CDC-ECM)
- A JTAG Pod
- Infrared USB Keyboard
- Sync: disk-on-key synchronization to Dropbox with the mbed system
- USB to I2C and to GPIO Bridge
- An MP3 Music Player
- CC2420 on FreeRTOS
- GDB for the LPC2148
- Transmitting high-speed ADC samples through UIP (IP over USB)
- A USB bootloader for for the LPC2148
- A CAN OBD Emulator
- EthVPN: Ethernet-virtualizing VPN device
- I-Feel: temperature sensor and infrared controller for an air conditioner

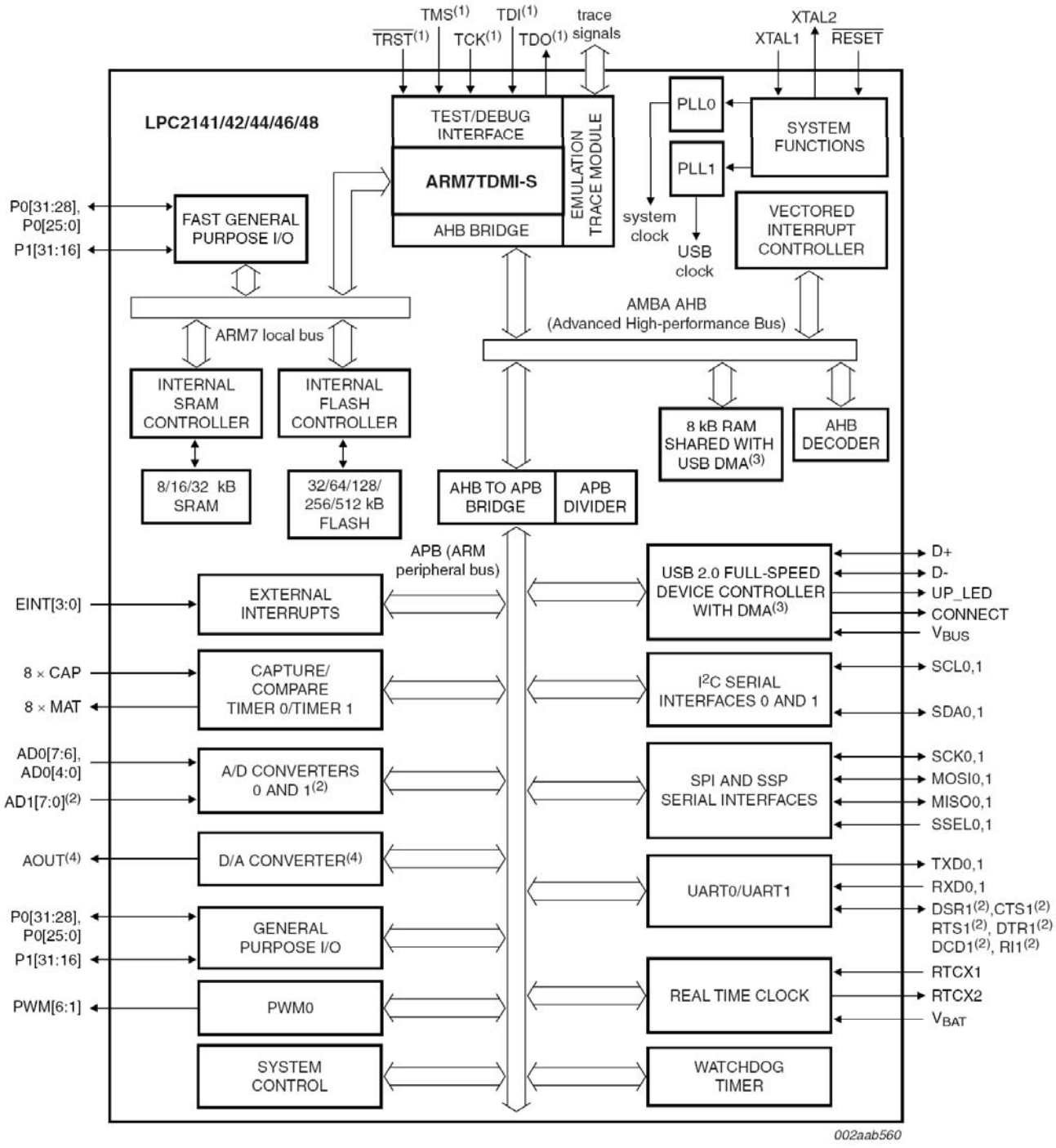
BASICS

Mechanics

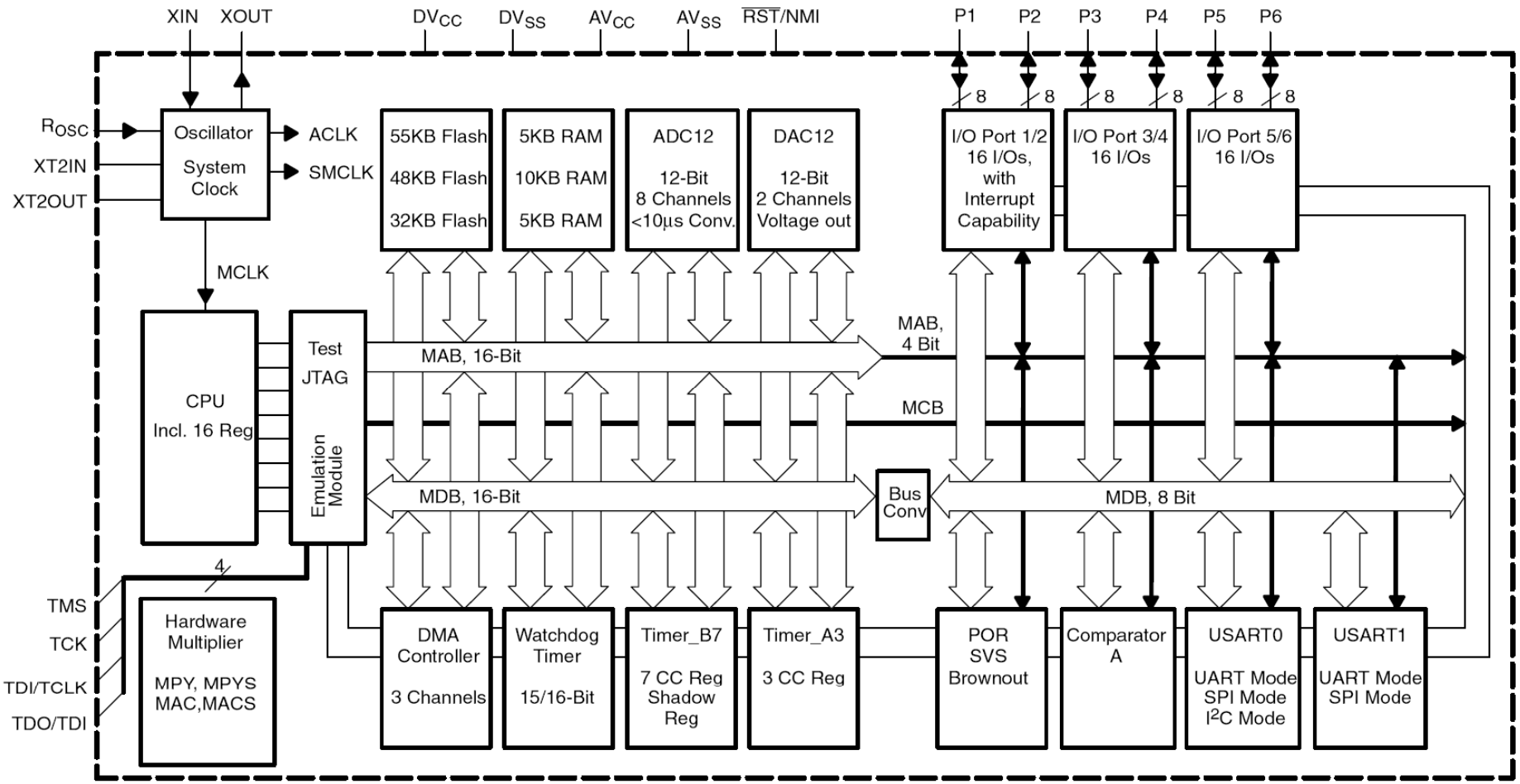

startup.S
assembler (as)
linker (ld)
prog.c
mcu.h
cross compiler (gcc)
assembler (as)
prog.elf
image converter
prog.hex
downloader (lpc21isp,...)
PC
special hardware
USB→serial
JTAG, proprietary
programming hardware
micro-controller

MCUs

Block diagram of the LPC2148

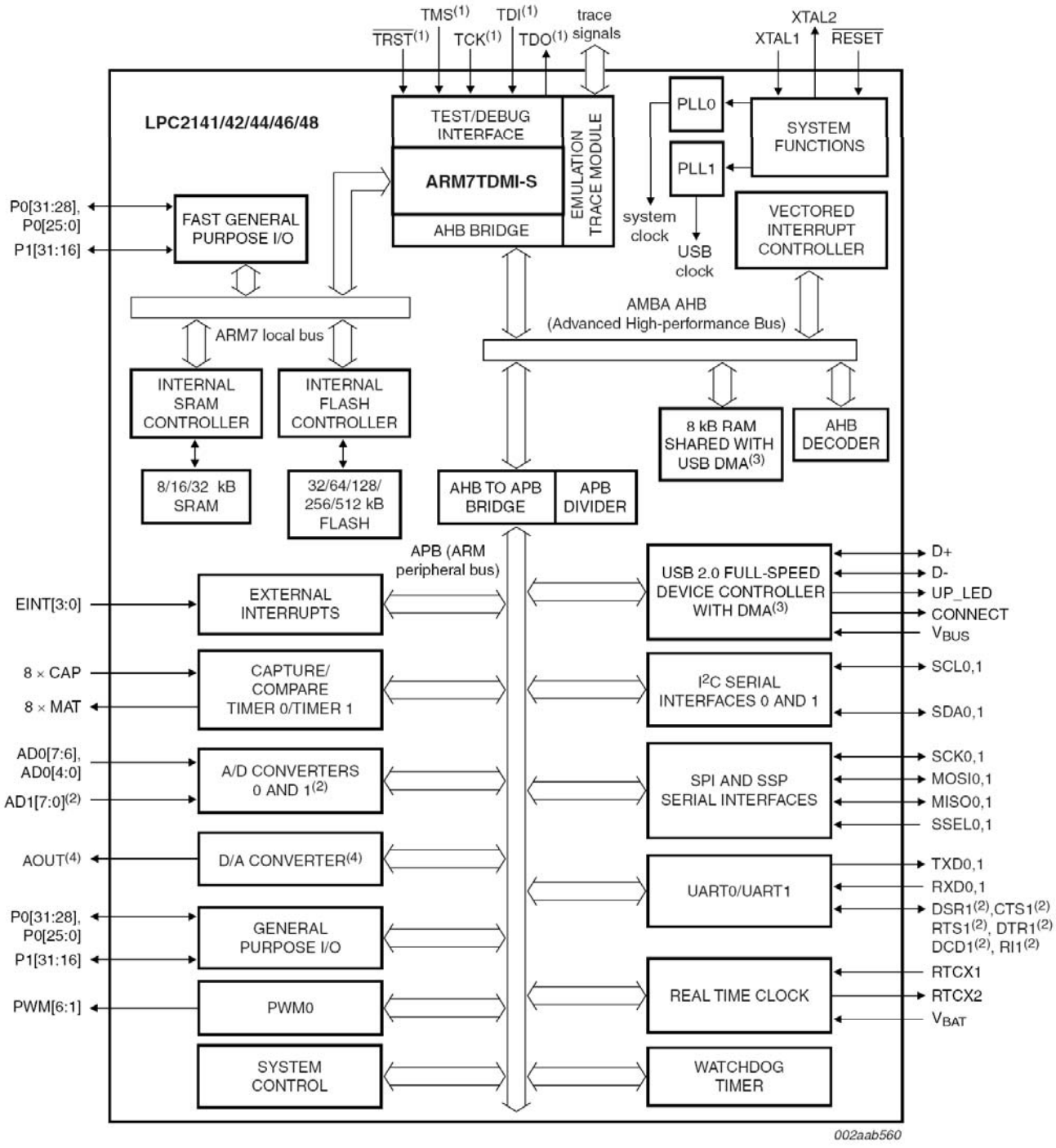


Microcontrollers: Block Diagram of the MSP430F1611

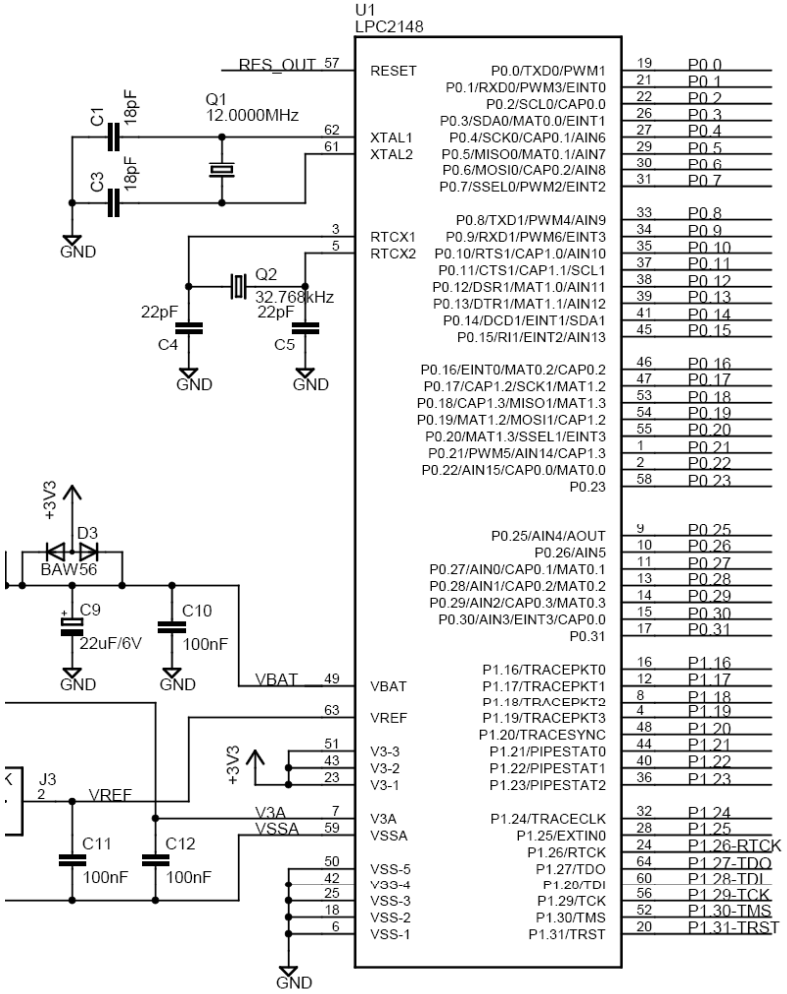


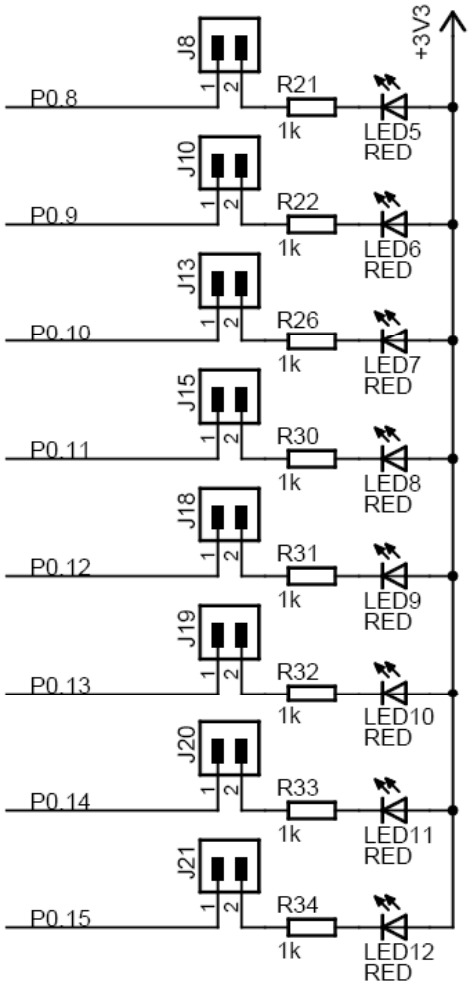
MCUs

Block diagram of the LPC2148



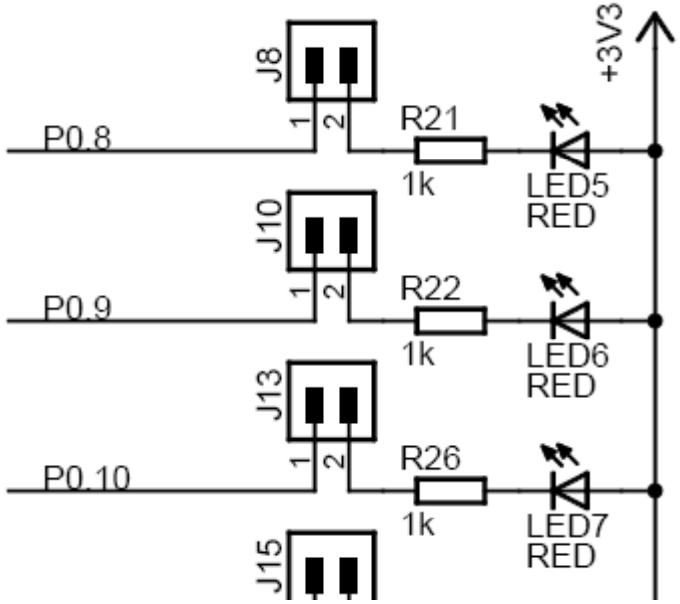
Building a simple program: Blinking an LED





Building a simple program: Blinking an LED

P0.8/TXD1/PWM4/AIN9	33	P0.8
P0.9/RXD1/PWM6/EINT3	34	P0.9
P0.10/RTS1/CAP1.0/AIN10	35	P0.10
P0.11/CTS1/CAP1.1/SCL1	37	P0.11
P0.12/DSR1/MAT1.0/AIN11	38	P0.12
P0.13/DTR1/MAT1.1/AIN12	39	P0.13
P0.14/DCD1/EINT1/SDA1	41	P0.14
P0.15/RI1/EINT2/AIN13	45	P0.15



Pin P0.10 is a digital I/O

Logic high→3.3V (MCU supply)

Logic low →0V (ground)

How do we toggle P0.10?

Toggling a Digital Output

- Consult reference (user) manual
- Go to *General Purpose I/O* (GPIO)
- Manual describes how *special function registers* (SFRs) interact with the hardware
- `IO0DIR` controls direction (input/output)
- Symbolic name for address `0xE0028008`
- `IO0PIN` controls/shows pin state (high/low)
- Header file defines symbolic names for SFRs:

```
#define IOPIN0 (*(volatile unsigned long*)(0xE0028008))
```

The Program

```
#include <lpc2000/io.h>

volatile int dummy;

const int delay = 100000;

void main(void) {
  int i;

  IODIR0 = BIT10; // BIT10==0b00..010000000000

  while (1) {
    for (i=0; i<delay; y++) dummy=1;
    IOPIN0 |= BIT10;
    for (i=0; i<delay; y++) dummy=1;
    IOPIN0 &= ~BIT10;
  }
}
```

The Program

```
#include <lpc2000/io.h>

volatile int dummy;

const int delay = 100000;

void main(void) {
  int i;

  IODIR0 = BIT10; // BIT10==0b00..010000000000

  while (1) {
    for (i=0; i<delay; y++) dummy=1;
    IOPIN0 |= BIT10;
    for (i=0; i<delay; y++) dummy=1;
    IOPIN0 &= ~BIT10;
  }
}
```

why volatile?

SFR is used like a variable (at a fixed address)

delay loop

on or off?

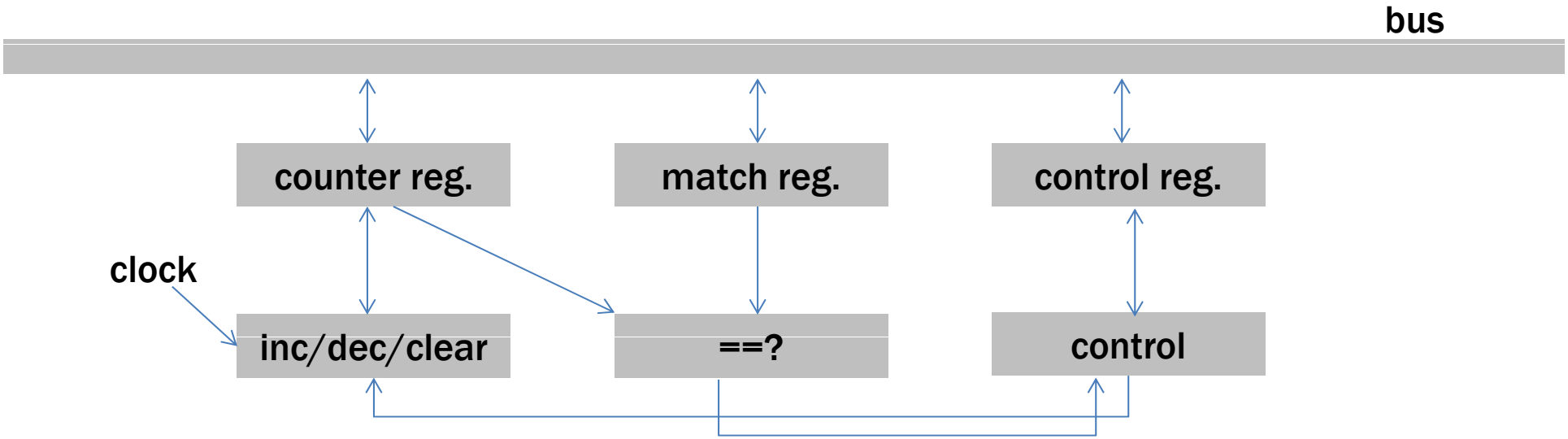
What Happens Before `Main`?

What Happens Before `Main`?

- Very little
- CPU starts with a fixed value in the program counter (0 in ARMs)
- In `Startup.S` address 0 is a branch to code
  - that sets up the stack pointer to end of RAM,
  - zeros unitialized global variables,
  - copies initial values of globals from flash to RAM,
  - and calls `main`
- What happens before `main` in Windows/Linux/MacOS?

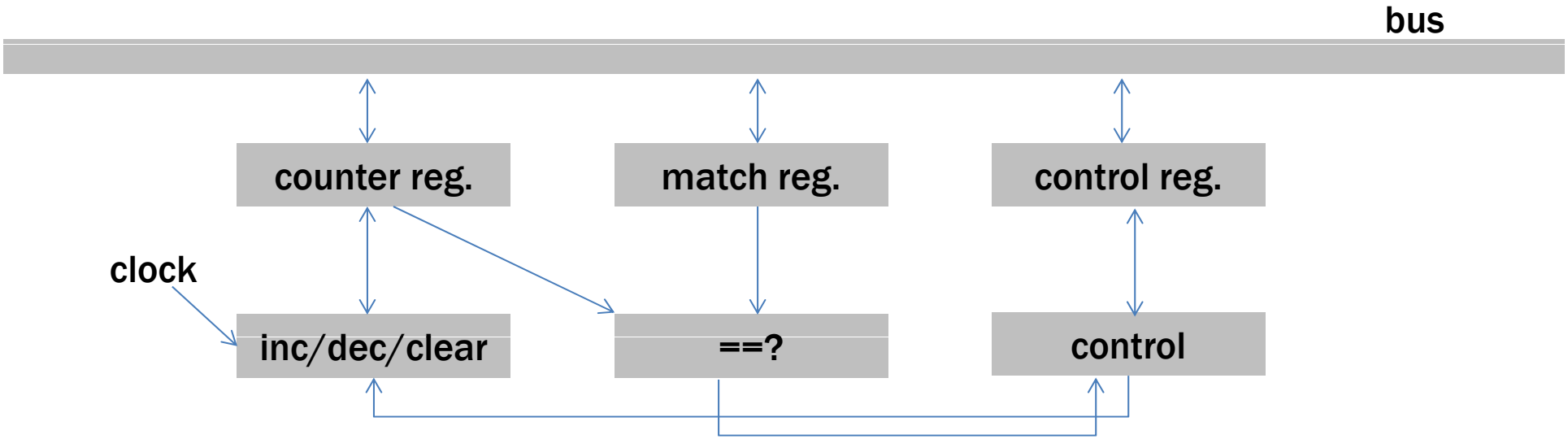
Timers/Counters

- Goal: replace the delay loop by an accurate & controllable mechanism
- We'll use a timer/counter peripheral



Timers/Counters

- On LPC2148, clock is PCLK, default freq on our board is 3MHz
- Strategy: match=3000000-1, configure to stop on match, start, restart in software



Timer/Counter: The Program

```
#include <lpc2000/io.h>

void main(void) {
  IODIR0 = BIT10;

  while (1) {
    T0TC = 0;            // zero the counter
    T0MR0 = 2999999;     // match register 0
    T0MCR = BIT2;        // stop on match
    T0TCR = BIT0;        // start the timer!
    while (T0TCR & BIT0); // while it's in run state

    IOPIN0 |= BIT10;
    ...                  // same delay mechanism as above
    IOPIN0 &= ~BIT10;
  }
}
```

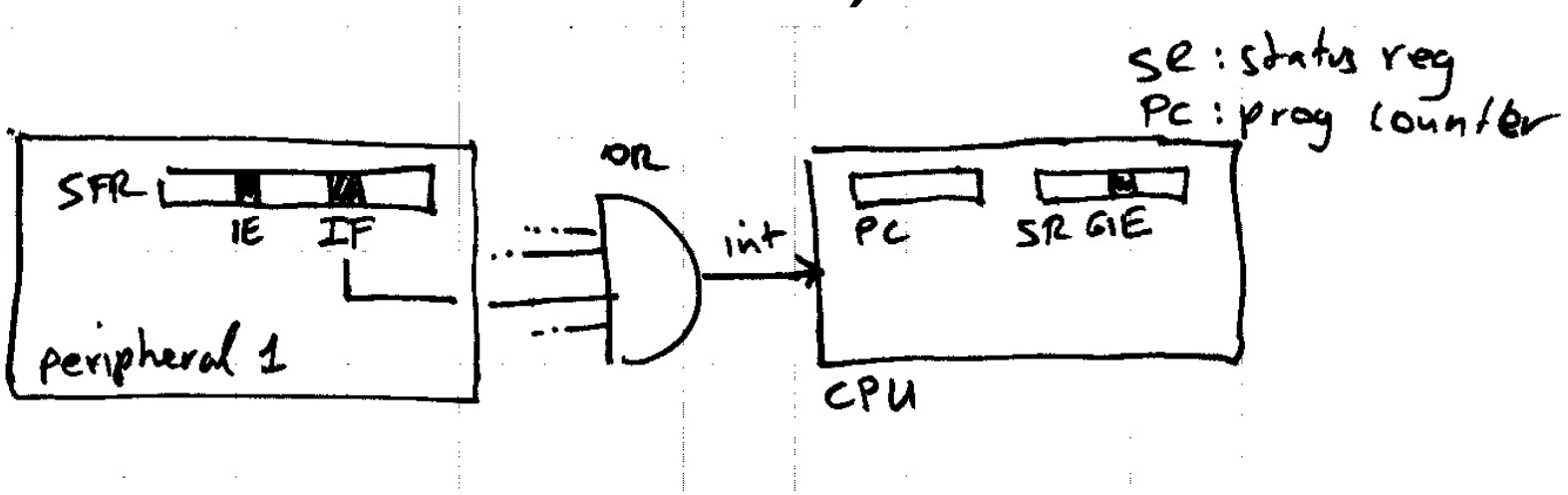
3 SFR Behaviors

- Configure the circuit
  - Bits in IODIR0 (dis)connect the output circuit from the physical pin
- Dual-ported memory
  - T0TC: we can assign a value and read it, and so can the timer (to increment)
- Trigger an action
  - Setting bit 0 in T0TCR starts the timer
  - Bit 1 (which we didn't use) clears T0TC

From Polling to Interrupts

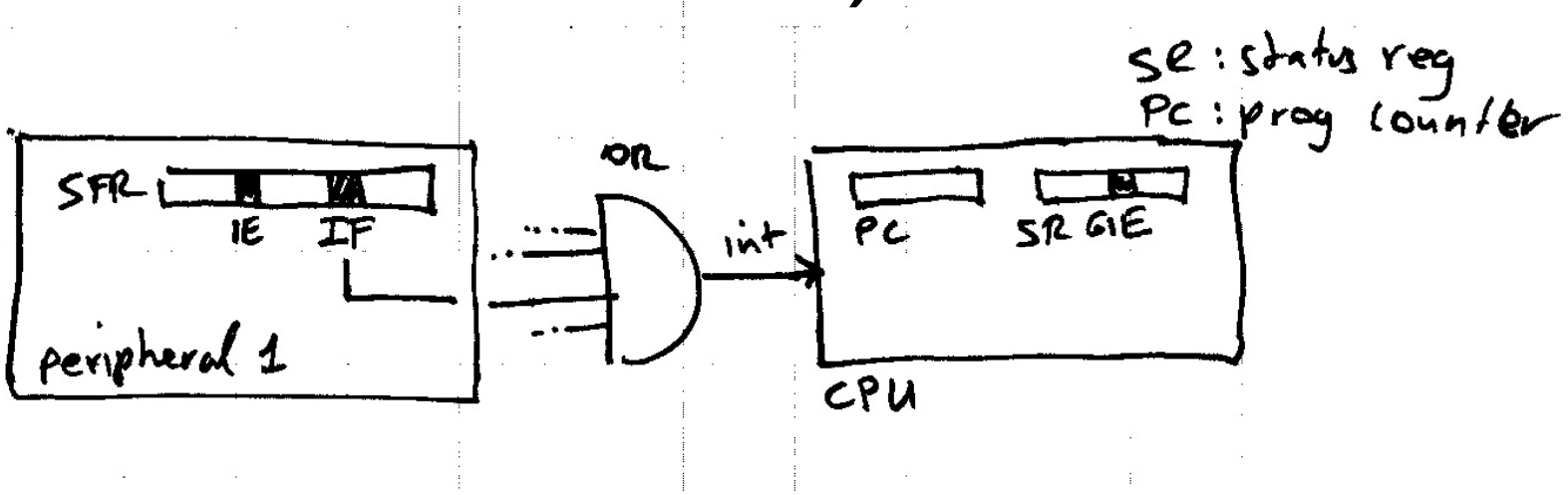
- Polling: check for a condition in a loop (or once in a while), take action when holds
- Wastes CPU resources, hard to modularize, and hard to reason about response times
- Better to ask the hardware to notify the program when condition holds
- E.g., ask the timer to invoke a function that resets the counter when a match occurs

Simple Interrupt Processing (e.g., on PIC16)



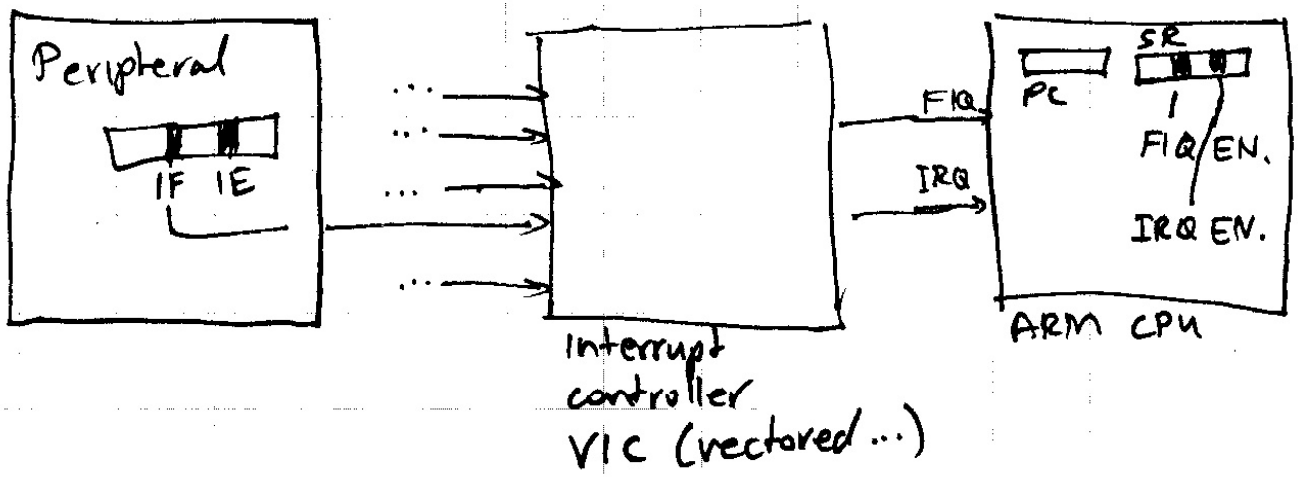
- Peripheral: at event, if IE (interrupt enable) then set IF (interrupt flag)

Simple Interrupt Processing (e.g., on PIC16)



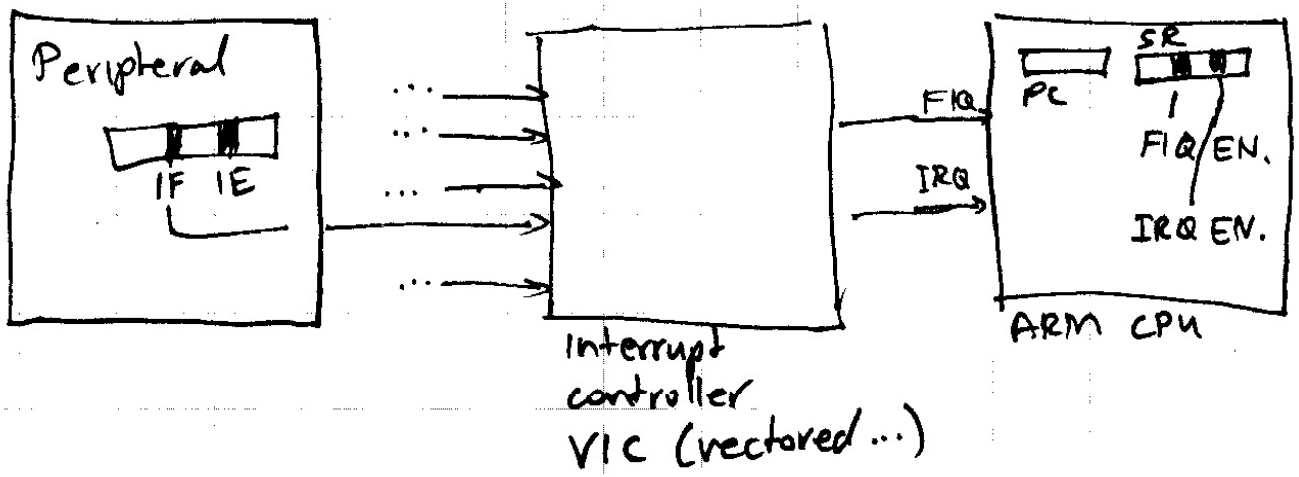
- CPU: before every instruction
  - If interrupt signal active & GIE (global IE), then
    - Clear GIE (avoids unintended recursion)
    - Save PC (on stack or in special register)
    - Load a fixed constant address into PC
    - Address is the start of an Interrupt Service Routine (ISR) or a branch to it

Interrupts on LPC2000 (Simplified)

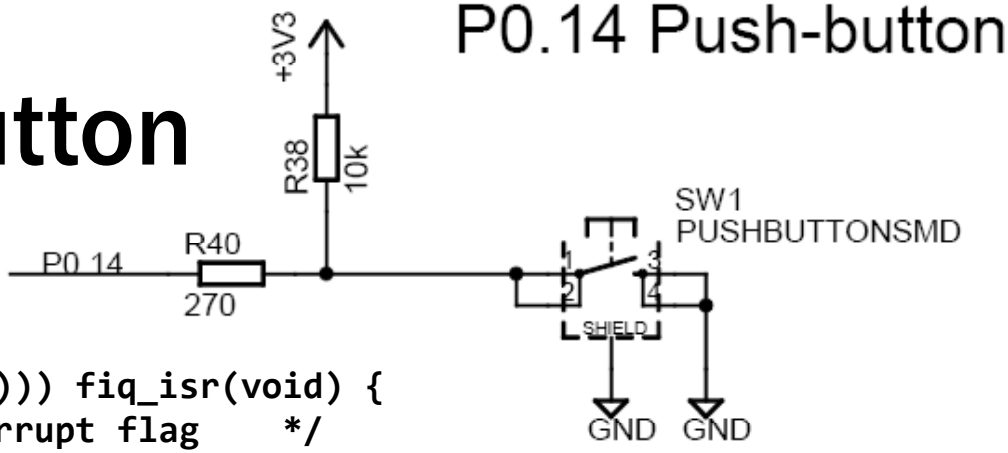


- VIC is programmed to set routing (→FIQ or →IRQ) and priority of IRQ ISRs
- FIQ ISR can interrupt IRQ ISR and is invoked faster

Interrupts on LPC2000 (Simplified)



More Interrupts: External from Button




```
#include <lpc2000/io.h>
#include <lpc2000/interrupt.h>

void __attribute__ ((interrupt("FIQ"))) fiq_isr(void) {
  EXTINT  = BIT1;  /* clear the interrupt flag    */

  if (IOPIN0 & BIT10) IOPIN0 &= ~BIT10;
  else                IOPIN0 |=  BIT10;  }

void main(void) {
  IODIR0 = BIT10;       /* pin P0.8 is output           */

  PINSEL0 = BIT29;      /* select EINT1 for P0.14      */
  EXTMODE = BIT1;       /* INT1 is edge sensitive      */
                        /* falling edge by default     */
  EXTINT  = BIT1;       /* clear the interrupt flag    */

  VICIntSelect = BIT15; /* EINT1 is the fast interrupt */
  VICIntEnable = BIT15; /* enable EINT1                */

  enable_interrupts();

  while (1) {}
}
```