YouLostIt:
Your privacy preserving battery-powered lost-device sensor

• Hardware
  – Accelerometer to detect when device is lost
  – LEDs to indicate who the lost owner is
    • (captured with smartphone video)
  – MCU that manages sensors, radios, and power
  – Bluetooth Low-Energy to tell a smartphone about lost device
    • Who owns it? How long has it been lost?
  – Flash memory to persistently store data
  – Clocks/Timers to keep track of time
  – Power from battery for several days to months.

• Software in a smartphone
  – Listen for nearby devices
  – Alert if the lost device goes out of range
YouLostIt:
Your privacy preserving battery-powered lost-device sensor

• Hardware
  – Accelerometer to detect when device is lost
  – LEDs to indicate who the lost owner is
    • (captured with smartphone video)
  – MCU that manages sensors, radios, and power
  – Bluetooth Low-Energy to tell a smartphone about lost device
    • Who owns it? How long has it been lost?
  – Flash memory to persistently store data
  – Clocks/Timers to keep track of time
  – Power from battery for several days to months.

• Software in a smartphone
  – Listen for nearby devices
  – Alert if the lost device goes out of range
Microcontroller VS Microprocessor

• A microcontroller is a small computer on a single integrated circuit containing a processor core, memory, and programmable input/output peripherals.

• A microprocessor incorporates the functions of a computer’s central processing unit (CPU) on a single integrated circuit.
Typical Microcontroller
How to choose MCU for our project?

• What metrics we need to consider?

  – Power consumption
    • E.g., we cannot afford high-power MCU because the power budget of the system requires lasting two years on one battery charge.

  – Clock frequency (speed that instructions are executed)
    • kHz is too slow...
    • 100MHz is over kill...

  – I/O
    • Lots of peripherals you can have:
      Image sensor, UART debugger, SD card, DAC, ADC, microphone, LED
How to choose MCU for our project?

• What metrics we need to consider?
  
  – Memory
    • We need to have sufficient memory to store:
      – Program (Non-volatile): Logic to read from sensors, communicate
      – Stack: Function calls are now expensive (no recursion)
      – Data: Constants (time periods), Sensor history, Communication state
        » We may need non-volatile data storage for data too (e.g., Flash)
  
  – Performance of internal peripherals
    • E.g., Speed of copying data from the sensor to the radio (DMA)
Memory Mapped IO

- Device registers mapped into regular address space
  
  - Use regular move (assignment) instructions to read/ write a device’s hardware “registers”
  
  - Use memory protection mechanism to protect device registers

- Memory Mapped I/O vs. Port I/O
  
  - Ports:
    - special I/O instructions are CPU dependent
  
  - Memory mapped:
    - memory protection mechanism allows greater flexibility than protected instructions
    - may use all memory reference instructions for I/O
    - cannot cache device registers (must be able to selectively disable caching)
    - I/O devices do not see the memory address - how to route only the right memory address onto slower peripheral buses (may initiate bridge at setup time to transfer required address areas)

- Intel Pentium uses a hybrid
  
  - address 640K to 1M is used for memory mapped I/O data buffers
  
  - I/O ports 0 to 64K is used for device control registers
GPIOs are the general digital I/O device

Each GPIO pin represents one bit in memory: if the pin is on it’s a 1, off it’s a 0. That bit can be an input or an output.

- GPIOs can be used to control lights (light on or off), but even more
  - Indicating that an event just happened
    - Interrupt the radio to tell it to transmit data
    - Interrupt the CPU to tell it a button was pressed
    - Read pin status to receive configuration messages
  - Debugging
    - Did this one part of my code actually execute?
    - Is the timer firing at the interval that I expect it to fire (connect GPIO to oscilloscope)?
    - Why using GPIO?
      - GPIO ops are lightweight
Topology of a GPIO pin

Functional Description
Figure 23-2. Overview of the PORT
Timers, why do we need them?

In the first project, what do we need timers for?

• Determining when to change LEDs
  – 20 Hz means change bits every 50 milliseconds
  – How to measure 50 ms?
  – Option 1: Use the timer hardware to let you know when 50 ms has passed.
  – Option 2: Count how many processor cycles it would take to equal 50 ms.
Timer: A peripheral for tracking time

The purpose of the prescaler is to allow the timer to be clocked at the rate a user desires.

Frequency depends on the attached oscillator device

Figure adapted from Prabal Dutta’s EE373 slides
How do interrupts work?

What is the benefit of having a separate controller for interrupts?
How a firmware programmer uses interrupts

1. Tell the peripheral which interrupts you want it to enable.

2. Tell the interrupt controller to enable that interrupt.

3. Tell the interrupt handler what that interrupt’s priority is.

4. Tell the processor where the interrupt handler is (function address).

5. When the interrupt handler fires, do the work then clear the int.
Even one LED can be critical for debugging

Think of an LED as a breakpoint (very efficient!)

• Turn it on before a line of code
  – Lets you know you reached that line

• Turn it off after a line of code
  – Lets you know you exited that code segment

• Toggle it each time a repeated event is called
  – This can show you visually how often the event happens
Simplistic View of Serial Port Operation

Transmitter

<table>
<thead>
<tr>
<th></th>
<th>n</th>
<th>n+1</th>
<th>n+2</th>
<th>n+3</th>
<th>n+4</th>
<th>n+5</th>
<th>n+6</th>
<th>n+7</th>
<th>n+8</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>n+1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>n+2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n+3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n+4</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n+5</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n+6</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n+7</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n+8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Receiver

<table>
<thead>
<tr>
<th></th>
<th>n</th>
<th>n+1</th>
<th>n+2</th>
<th>n+3</th>
<th>n+4</th>
<th>n+5</th>
<th>n+6</th>
<th>n+7</th>
<th>n+8</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>n+1</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n+2</td>
<td>6</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n+3</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n+4</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n+5</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n+6</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n+7</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>n+8</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

Interrupt raised when Transmitter (Tx) is empty
⇒ Byte has been transmitted
and next byte ready for loading

Interrupt raised when Receiver (Rx) is full
⇒ Byte has been received
and is ready for reading
UART Protocol

• Each character is sent as
  – a logic low **start** bit
  – a configurable number of data bits (usually 7 or 8, sometimes 5)
  – an optional parity bit
  – one or more logic high **stop** bits
  – with a particular bit timing (“baud”)

![ UART Protocol Diagram ]
SPI Protocol

• Wires:
  – Master Out Slave In (MOSI)
  – Master In Slave Out (MISO)
  – System Clock (SCLK)
  – Slave Select 1…N

• Master Set Slave Select low

• Master Generates Clock

• Shift registers shift in and out data
I2C Details

• Each I2C device recognized by a unique address

• Each I2C device can be either a transmitter or receiver

• I2C devices can be primarys or secondarys for a data transfer
  – primary (usually a microcontroller): Initiates a data transfer on the bus, generates the clock signals to permit that transfer, and terminates the transfer
  – secondary: Any device addressed by the primary at that time