Jay Rathod
T-127

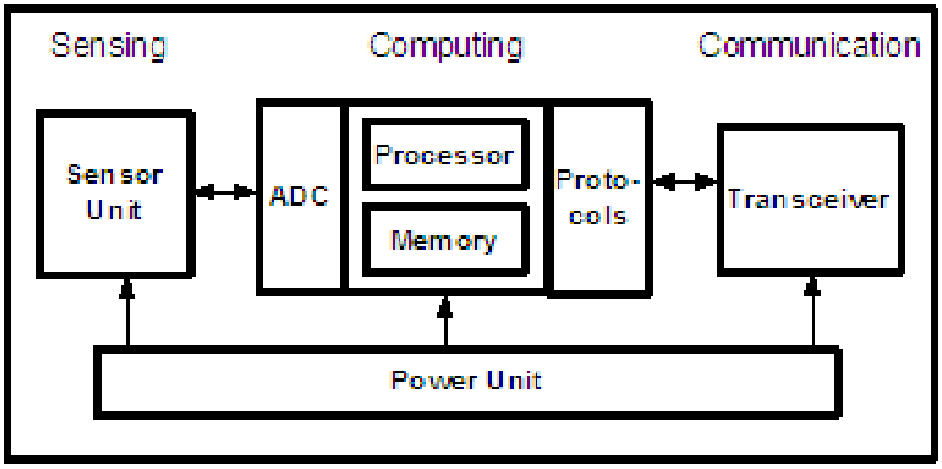
Wireless

Practical 1

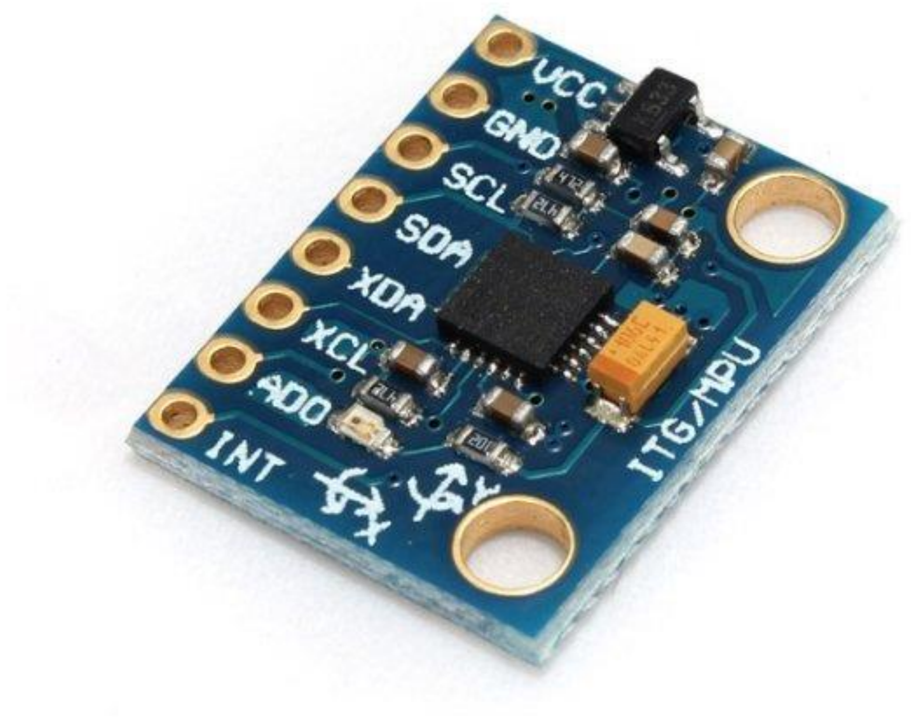
Aim: Understanding the Sensor Node Hardware. **(Accelerometer Sensor)**

Description:

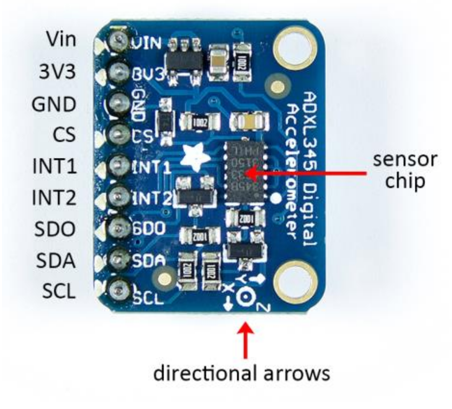
1. A sensor node, also known as a mote (chiefly in North America), is a node in a sensor network that is capable **of performing some processing, gathering sensory information and communicating** with other connected nodes in the network.



2. Accelerometer sensors are **ICs that measure acceleration**, which is the change in speed (velocity) per unit time. Measuring acceleration makes it possible to obtain information such as object inclination and vibration. ... g is also used as a unit for acceleration, relative to standard gravity (1g = 9.80665m/s^2).



3. Components



The directional arrows indicate the direction of each sensor axis (x, y, and z) in relation to the physical board. The z axis extends up and down, perpendicular to the x and y axes.

4. **Pin Configurations**

- Vin – Connects to a 5 volt power source.
- 3V3 – The ADXL345 can be powered with 3.3 volts or 5 volts. This pin is where the 3.3 volt power source would connect.
- GND – Connects to ground.
- CS – Chip select pin for SPI communication.
- INT1 – Hardware interrupt pin 1.
- INT2 – Hardware interrupt pin 2.
- SDO – Serial data output pin. Doubles as the MISO pin for SPI communication.
- SDA – SDA pin for I2C communication. Doubles as the MOSI pin for SPI communication.
- SCL – SCL pin for I2C communication. Doubles as the SCLK pin for SPI.

5. **Working Principle**

Acceleration is the rate of increase or decrease of velocity:

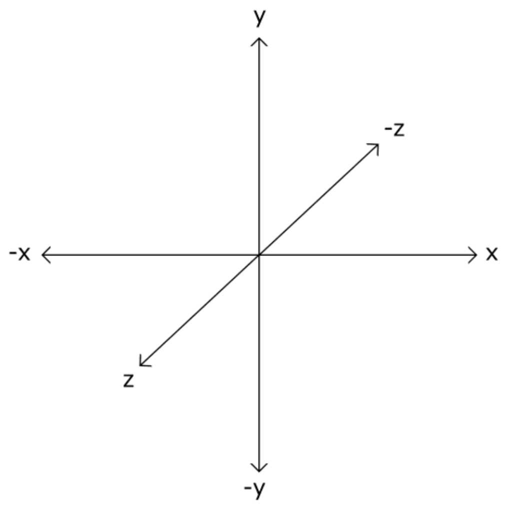
$$a = \frac{v}{t}$$

If you drive in a car and step on the gas, the car has a positive acceleration. If you step on the brakes, the car has a negative acceleration. This is known as *dynamic acceleration*.

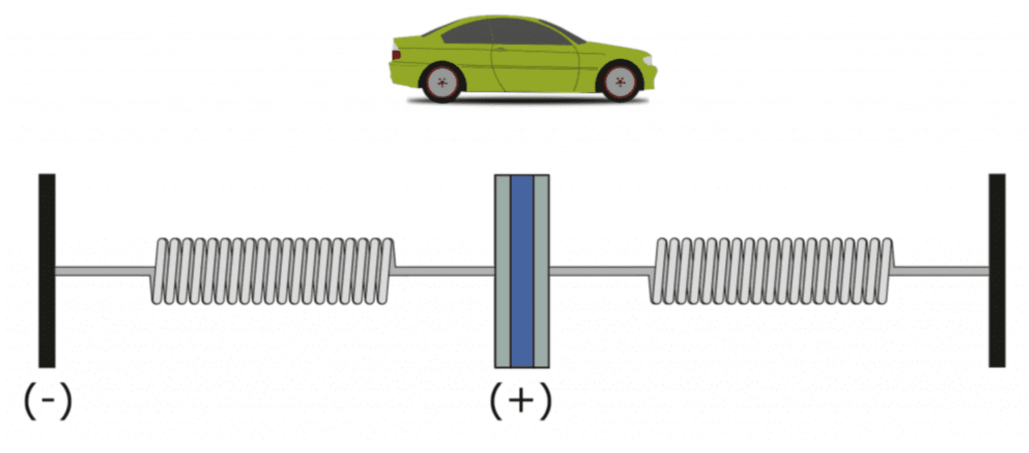
Static acceleration is caused by forces like gravity. Accelerometers measure both static acceleration and dynamic acceleration.

Two common units of acceleration are meters per second squared (m/s^2) and g's. One g is defined as the rate of acceleration of gravity, which is 9.8 m/s^2.

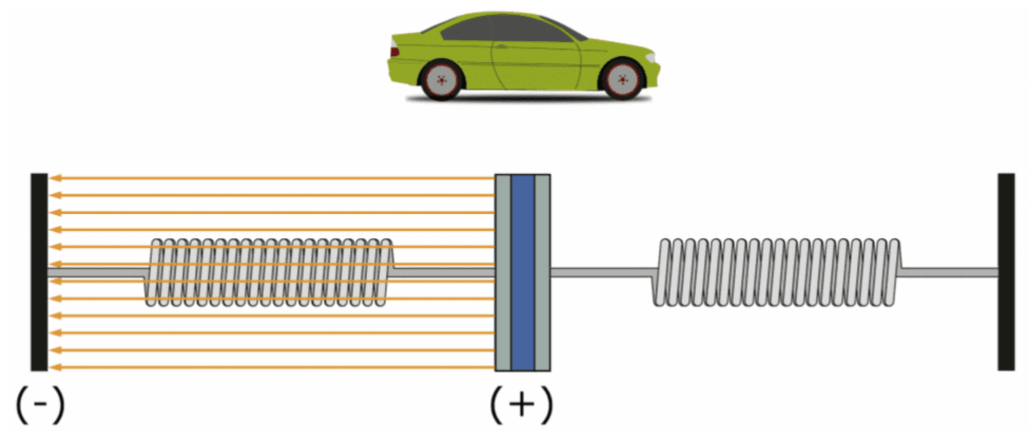
The ADXL345 outputs separate acceleration measurements for each axis, x, y, and z:



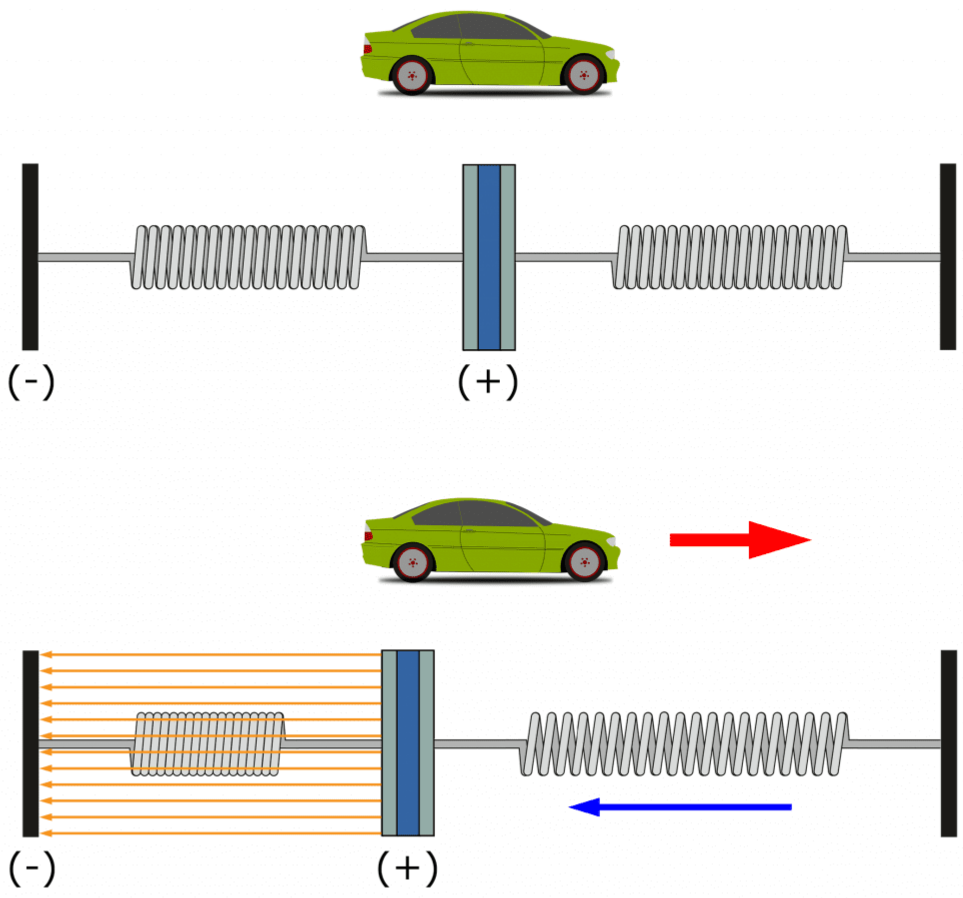
The ADXL345 measures acceleration by detecting changes in capacitance. Along each axis there is a tiny plate suspended between two micro-springs that can move back and forth:



The mobile plate and the fixed plate are charged, so an electric field is formed between them:

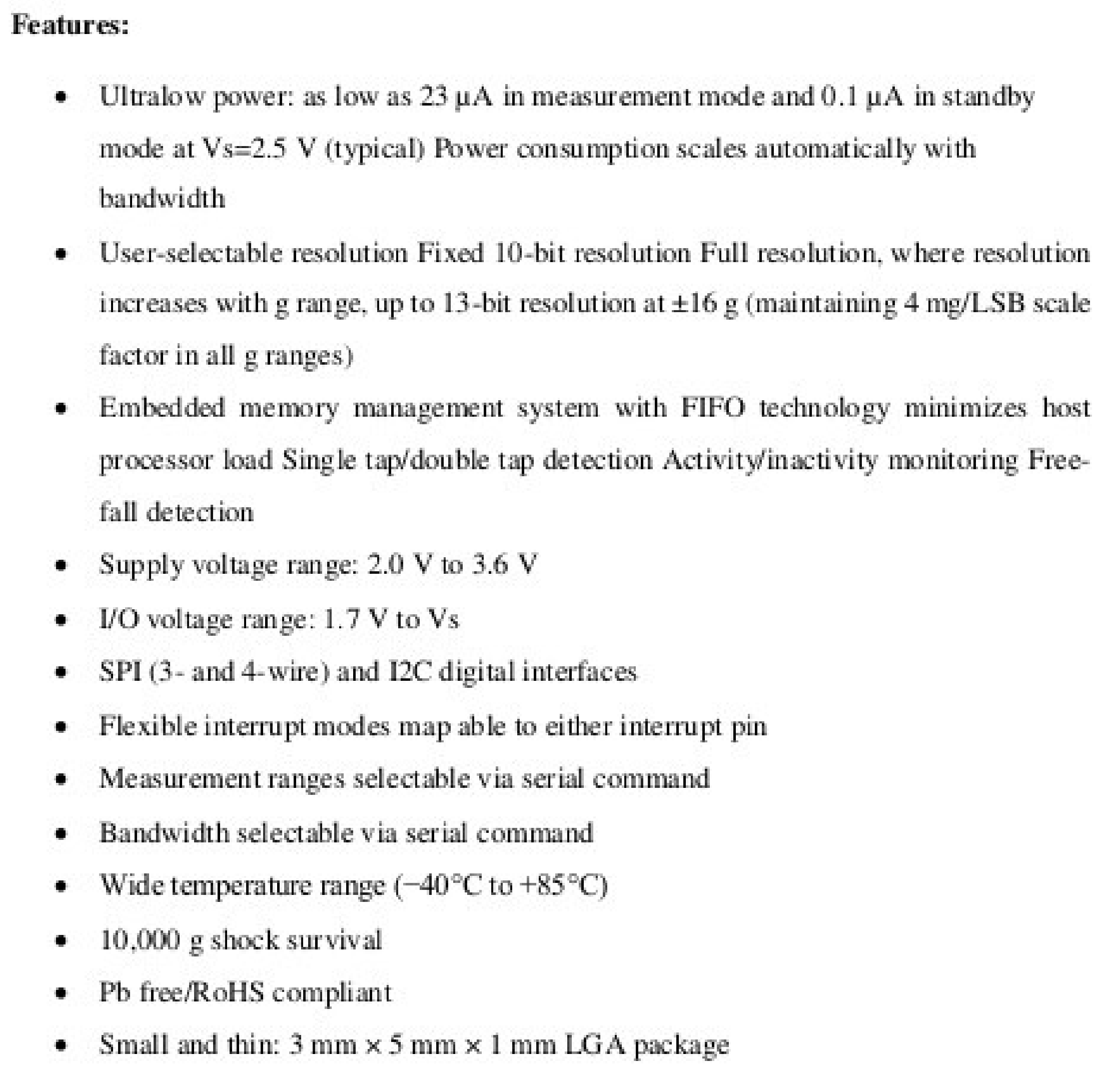


When the accelerometer is at rest, the electric field between the plates is constant. When the sensor accelerates, the mobile plate moves, and the distance between the plates changes.



Capacitance is a function of the distance between two charged plates, so when the distance between the plates changes, the electric field between the plates also changes. The sensor measures this change in capacitance and calculates an acceleration value.

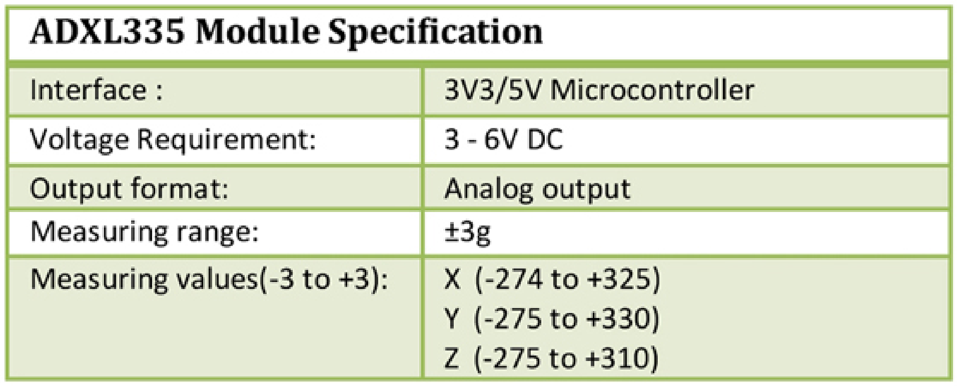
6. Features



Features:

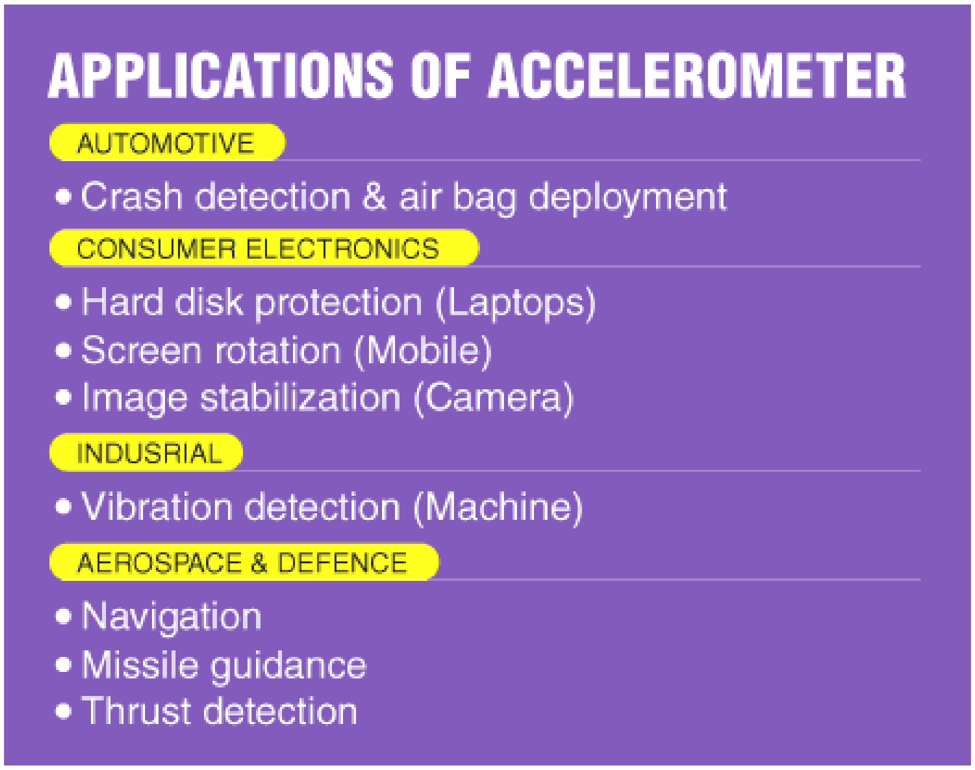
- Ultralow power: as low as 23 µA in measurement mode and 0.1 µA in standby mode at Vs=2.5 V (typical) Power consumption scales automatically with bandwidth
- User-selectable resolution Fixed 10-bit resolution Full resolution, where resolution increases with g range, up to 13-bit resolution at ±16 g (maintaining 4 mg/LSB scale factor in all g ranges)
- Embedded memory management system with FIFO technology minimizes host processor load Single tap/double tap detection Activity/inactivity monitoring Free-fall detection
- Supply voltage range: 2.0 V to 3.6 V
- I/O voltage range: 1.7 V to Vs
- SPI (3- and 4-wire) and I2C digital interfaces
- Flexible interrupt modes map able to either interrupt pin
- Measurement ranges selectable via serial command
- Bandwidth selectable via serial command
- Wide temperature range (−40°C to +85°C)
- 10,000 g shock survival
- Pb free/RoHS compliant
- Small and thin: 3 mm × 5 mm × 1 mm LGA package

7. Specifications



ADXL335 Module Specification	
Interface :	3V3/5V Microcontroller
Voltage Requirement:	3 - 6V DC
Output format:	Analog output
Measuring range:	±3g
Measuring values(-3 to +3):	X (-274 to +325) Y (-275 to +330) Z (-275 to +310)

8. Applications



APPLICATIONS OF ACCELEROMETER

AUTOMOTIVE

- Crash detection & air bag deployment

CONSUMER ELECTRONICS

- Hard disk protection (Laptops)
- Screen rotation (Mobile)
- Image stabilization (Camera)

INDUSRIAL

- Vibration detection (Machine)

AEROSPACE & DEFENCE

- Navigation
- Missile guidance
- Thrust detection

Jay Rathod
T-127

Practical 2

Aim: Exploring and understanding TinyOS computational concepts:-
Events, Commands and Task.
- nesC model
- nesC Components

TinyOS:

TinyOS is an operating system for sensor nodes 1. Open source project with a strong academic background 2. Hardware drivers, libraries, tools, compiler

TinyOS applications are written in nesC

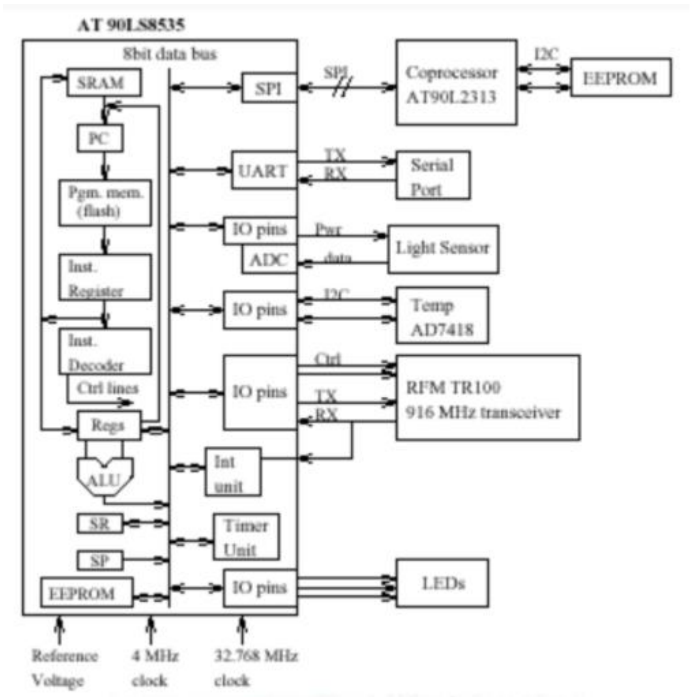
1. C dialect with extra features

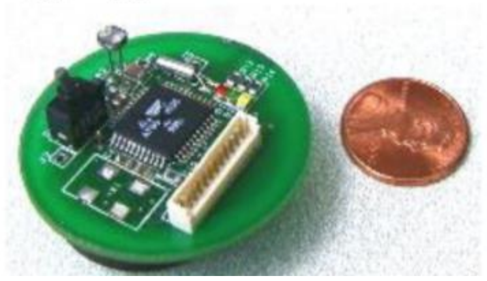
2. nesC compiler converts your application into plain C code

Networked Sensor Characteristics:

Small physical size and low power consumption. Concurrency-intensive operation. Limited physical parallelism and controller hierarchy. Diversity in Design and Usage. Robust Operation.

Photograph and schematic for theNetwork Sensor:





<u>The Solution: TinyOS:</u>

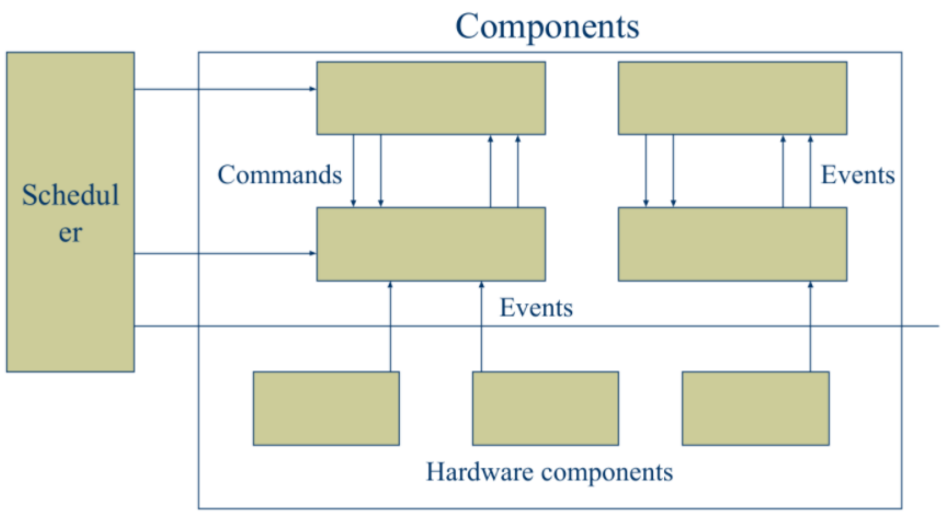
A micro threaded OS that draws on previous work done for lightweight thread support, and efficient network interfaces.

Two level scheduling structure:

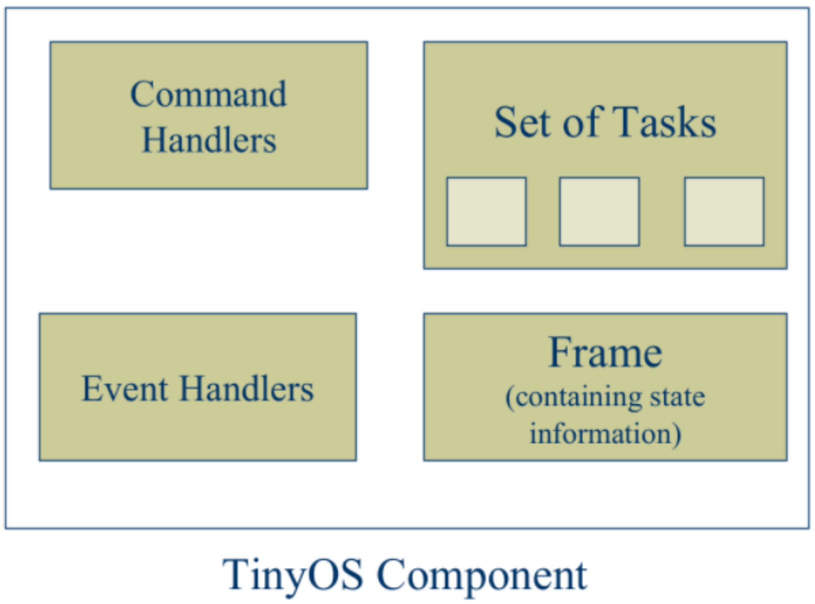
1. Long running tasks that can be interrupted by hardware events.

2. Small, tightly integrated design that allows crossover of software components into hardware.

<u>TinyOS - Design:</u>



Structure of a Component:



TinyOS Component Model:

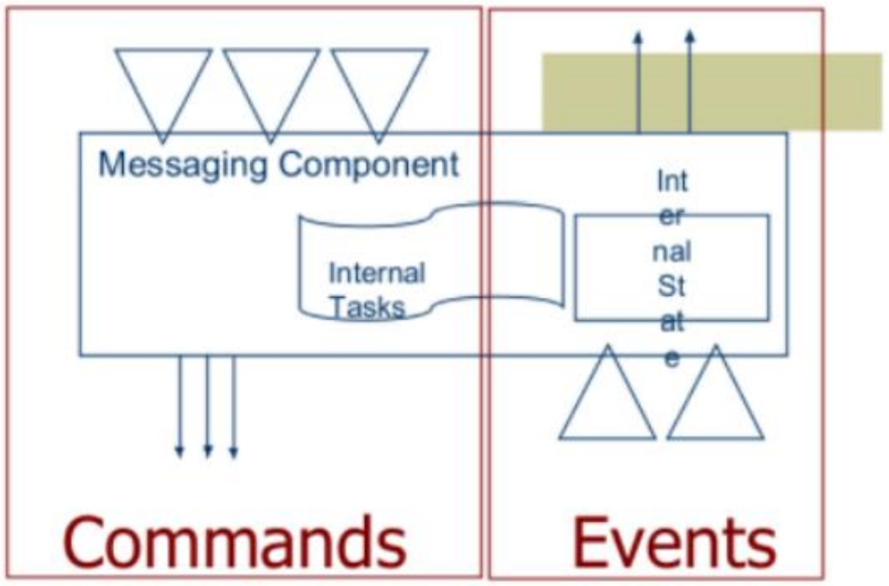
Component has:

1. Frame (storage)
2. Tasks (computation)

3. Command and Event Interface

To facilitate modularity, each component declares the commands it uses and the events it signals.

Statically allocated, fixed sized frames allow us to know the memory requirements of a component at compile time, and avoid the overhead associated with dynamic allocation.



Tasks:

1. Perform the primary computation work
2. Atomic with respect to other tasks, and run to completion, but can be preempted by events
3. Allow the OS to allocate a single stack assigned to the currently executing task
4. Call lower level commands
5. Signal higher level events

6. Schedule other tasks within a component

7. Simulate concurrency using events

Commands:

1. Non-blocking requests to lower level components

2. Deposit request parameters into a component’s frame, and post a task for later execution

3. Can also invoke lower level commands, but cannot block

4. To avoid cycles, commands cannot signal events

5. Return status to the caller

Events:

1. Event handlers deal with hardware events (interrupts) directly or indirectly

2. Deposit information into a frame

3. Post tasks

4. Signal higher level events

5. Call lower level commands

Sample Application:

1. The sample application is consisting of a number of sensors distributed within a localized area

2. Monitor temperature and light conditions

3. Periodically transmit measurements to a base station Sensors can forward data for other sensors that are out of range of the base station

4. Dynamically determine the correct routing topology for the network

NesC/TinyOS Programming Model:

Programs are built out of components

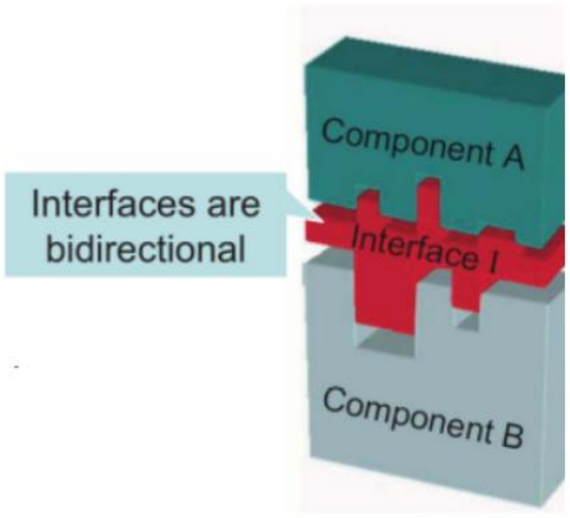
• Two types of components:

Modules: Implement program logic

Configurations: Wire components together

• Components use and provide interfaces

• Components are wired together by connecting interface users with interface providers



Programming Model:

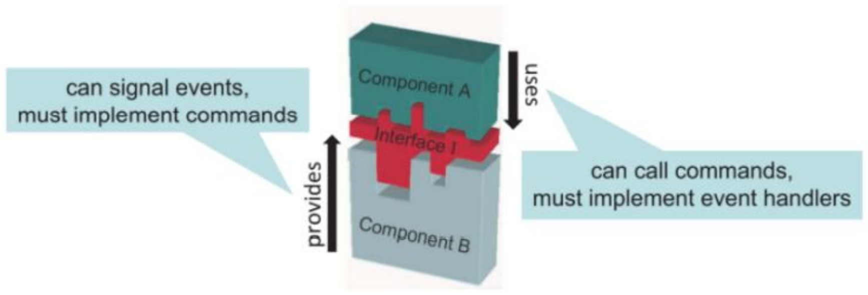
• Interfaces contain definitions of

• Commands

• Events

• Components implement the event handlers they use and the commands they provide

Jay Rathod
T-127



Concurrency Model:

Coarse-grained concurrency only Implemented via tasks

• Tasks are executed sequentially by the TinyOS scheduler no threads Atomic with respect to other tasks (single threaded) Longer background processing jobs

• Events (interrupts) Time critical Preempt tasks Short duration (hand off computation to tasks if necessary)

Memory Model:

• Static memory allocation

No heap (malloc)

No function pointers

• Global variables

One namespace per component

• Local variables

Declared within a function

Saved on the stack

Jay Rathod
T-127

nesC Programming Language:

Components:

Implementation: -

module: C behavior

Configuration: select and wire

Interfaces: provides interface requires interface

nesC much nicer to use than TinyOS:

1. produces smaller code than TinyOS.
2. get rid of the macros for frames, commands, events
3. eliminate wiring error through type checking
4. simplify wiring by using interfaces
5. increase modularity by separation of interfaces and modules