The CMU FireFly Platform

Programming FireFly and the Nano-RK Sensor OS

Lecture #3

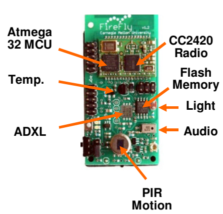
Previous Lecture

- Wireless Sensor Networks: Introduction and Applications

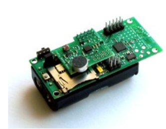
Outline of Today's Lecture

- **FireFly Hardware**
  - Atmel Processor Features
  - Chipcon Radio
  - FireFly Node
  - FireFly Programmer
- **Operating System**
  - Nano-RK
  - Configuring and Using Tasks
  - Tips and Tricks
- **Lab Descriptions**

FireFly History



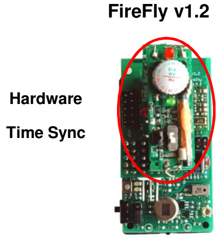
FireFly v2.0



FireFly v2.2

FireFly 3 Core Module

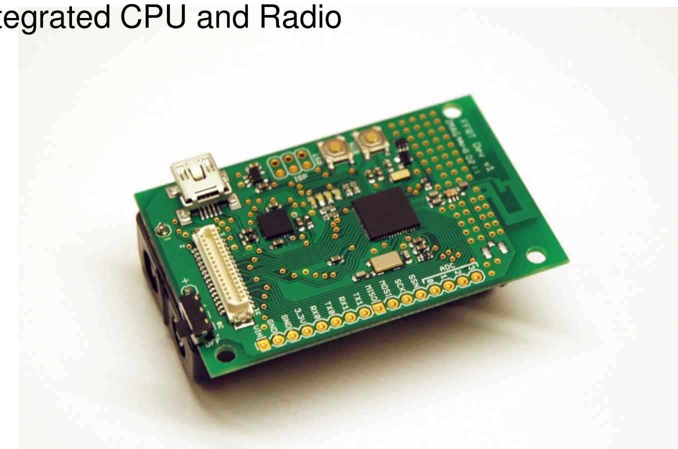
FireFly v1.2



FireFly 3 Dev Board

FireFly v3 Overview

- ATmega128RFA1 (System-On-Chip SoC)
- Integrated CPU and Radio



ATmega128rfa1 Features

- Integrated IEEE 802.15.4 Radio (more later)
- 1 I2C controller (2-wire serial protocol, Philips)
- 8-bit Harvard Architecture (16-bit address space)
- RISC instruction set
- 2 UARTs (RS-232 115,200 baud)
- 1 SPI (4-wire serial protocol, Motorola)
- 8-channel 10-bit ADC
- 2 8-bit Timers / Counters
- 4 16-bit Timers / Counters
- 6 full 8-bit GPIO ports
- 32 8-bit Registers
- 64 Control Registers

Atmel AVR CPU Core

- No Cache
- No Pre-Fetch
- No Branch Predictor
- No Memory Management Unit
- Yes, Pipeline (Hooray 1962!)

- Little RAM
- Reasonable flash

AVR On Chip Memory

- **SRAM (16K)**
  - Holds Program Data etc.
  - Cleared on Power Down

- **Flash Memory (128K)**
  - Holds Program Code
  - Does not get cleared on power down, but is slow for a program to write into (must write large pages)
  - Limited write-cycle lifetime

- **EEPROM (4K)**
  - Does not get cleared on power down
  - Easy read / write at runtime
    - Limited lifetime but typically better than flash memory

- **Fuses (a few bytes of non-volatile memory)**
  - Holds CPU Hardware Configuration
  - CPU *cannot* access this data

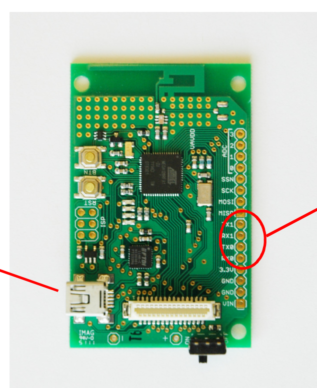
Data Memory

- **32 8-bit Registers**
  – Compiler (or you) use these for many instructions that only operate on registers
- **64 8-bit I/O Registers**
  – Located at back of CPU datasheet
  – Control UART parameters
  – Set / Read Pin Values
  – Configure Timers
- **Internal SRAM**
  – This is memory that must be loaded and written to and from registers
  – This is slower than directly accessing a register
  – Program Data, Stack and Heap live here

Address (HEX)	
0 - 1F	32 Registers
20 - 5F	64 I/O Registers
60 - 1FF	416 External I/O Registers
200	Internal SRAM (8192 x 8)
21FF	
2200	~~External SRAM (0 - 64K x 8)~~
FFFF	

AVR UART (Universal Asynchronous Receiver / Transmitter)

USB to Serial on UART 0



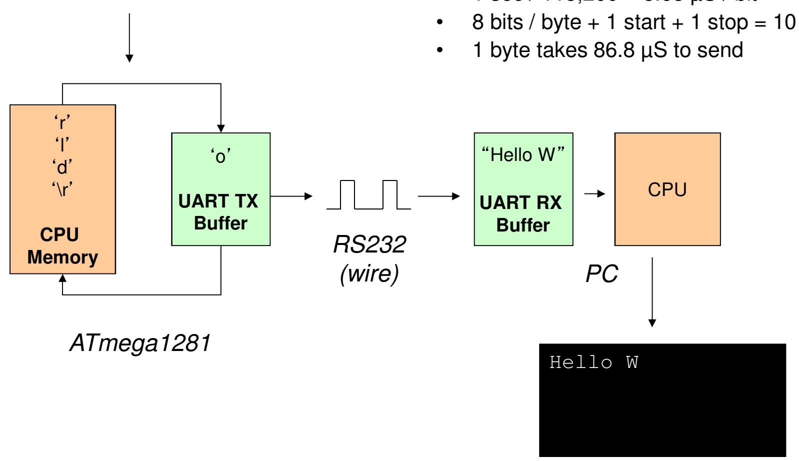
2 TTL UARTs

AVR UART

`printf( “Hello World…\r\n” );`

Source Code

- 1-byte TX and RX Buffer
- 115,200 baud (bits / second)
- 1 sec / 115,200 = 8.68 µS / bit
- 8 bits / byte + 1 start + 1 stop = 10 bits
- 1 byte takes 86.8 µS to send



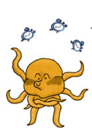
Terminal Program (minicom / hyperterminal)

Nano-RK UART

Example of printing a string and then a variable.

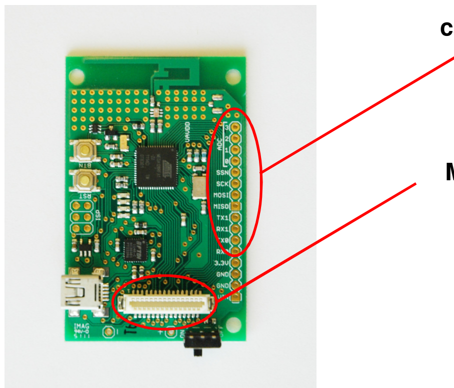
Use `nrk_kprintf()` whenever possible to save stack memory.

```
…
int8_t x;
  x = 5;
  nrk_kprintf(PSTR(“This is a constant string…"));
  printf( “x=%d",x);
…
```



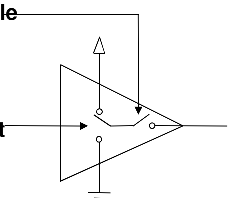
`PSTR()` indicates that this is a constant string stored in flash.
`nrk_kprint()` <u>only</u> operates on constant `PSTR()` strings.

AVR GPIO
(General-Purpose Input Output)



General-Purpose Input-Output (GPIO)

- Tri-state buffer port
- Memory-Mapped IO
- Control Registers
  - Data Direction Register (DDR, TRIS)
  - Internal Pull-Up Register (PLP)
  - Reading / Writing to the Port



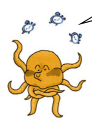
Each GPIO port can be configured as input or output on a bit-by-bit basis.

Input can act as a high-impedance state to allow other devices to control the line level.

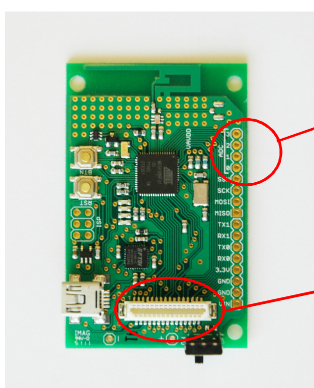
Nano-RK GPIO

Example of how to poll for the state of the button…

```
…
nrk_gpio_direction(NRK_BUTTON, NRK_INPUT);
while (1){
      if (nrk_gpio_get(NRK_BUTTON) == 0) break;
      else nrk_kprintf( PSTR("waiting…\r\n"));
}
nrk_led_set( BLUE_LED );
…
```



AVR Analog to Digital Converter

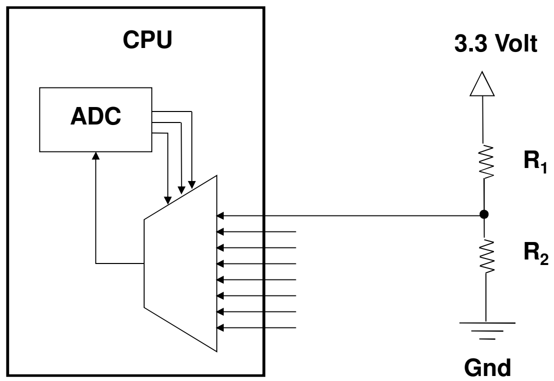


4 ADC Inputs

All 8 ADC Inputs

AVR Analog to Digital Converter (ADC)

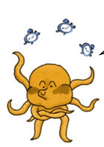
- Converts an Analog signal to a Digital signal
- 8 Channels at 10-bit Resolution
- Sampling Rate (up to 15 kSPS)



Nano-RK ADC

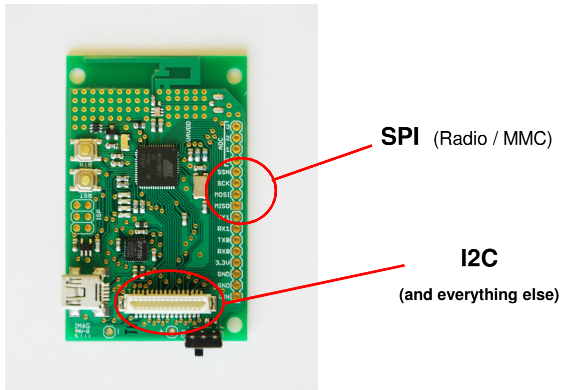
Example of how to read the ADC using the Nano-RK ADC device driver

```
int8_t fd, buf, val;

  fd = nrk_open(ADC_DEV_MANAGER,READ);
  if (fd == NRK_ERROR)
        nrk_kprintf(PSTR("Failed to open adc driver\r\n"));
  val = nrk_read(fd, &buf, 1);
  printf( "ADC value = %d", buf);
```

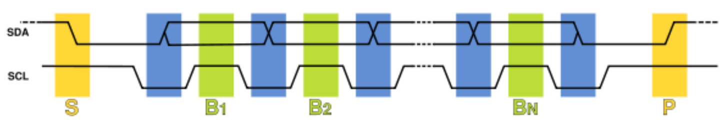


AVR SPI and I2C

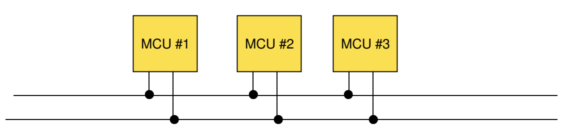
SPI: Serial Peripheral Interface bus
I2C: really I²C → Inter-Integrated Circuit



I²C (Inter-Integrated Circuit)

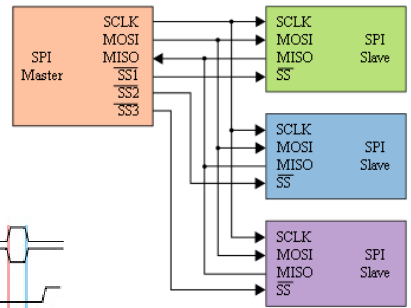


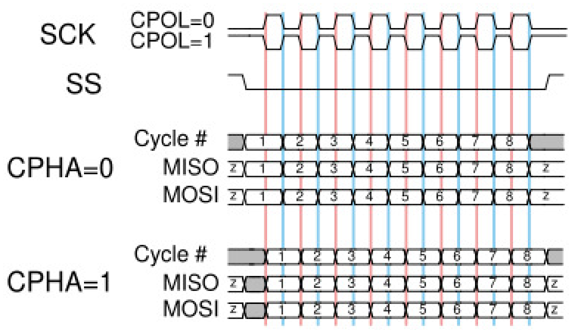
- Multi-drop 2-wire serial bus protocol
- Designed by Philips
- Two Bi-directional Open-Drain Lines
- 400 kbits / sec (up to 3.4Mbits / sec high-speed mode)



SPI (Serial Peripheral Interface)

- 4-Wire Serial Bus from Motorola
- Up to 1Mbits / sec
- Bidirectional Data

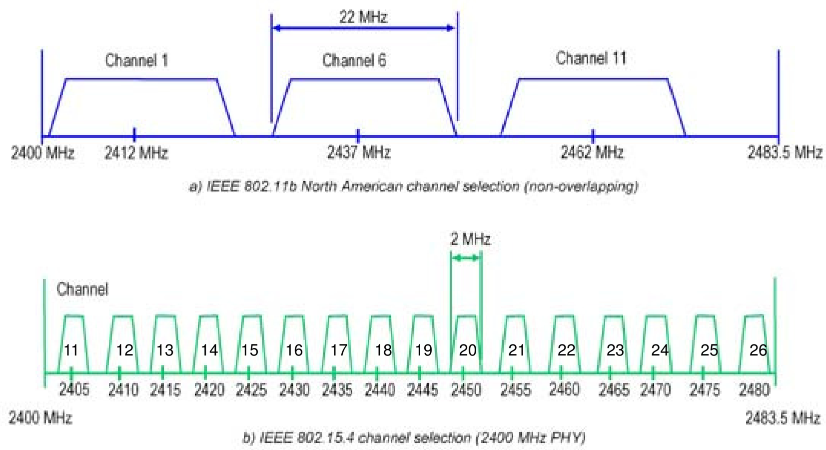




Radio (integrated AT86RF230)

- IEEE 802.15.4 Radio
  - Physical / MAC layer of Zigbee
    - Zigbee includes a higher-level protocol
- 2.4 GHz (ISM band)
- 250 Kbps (Burst Data Rate)
- 16 Channels (11-26) in 5MHz Steps
- Transmits > 100 meters line of sight
- 0.18μM CMOS Process

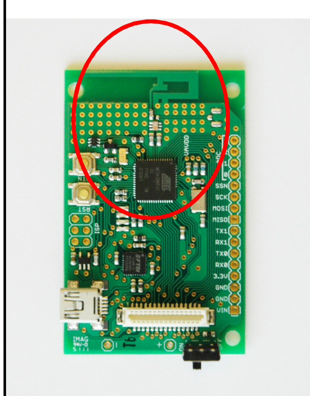
802.15.4 vs 802.11.b

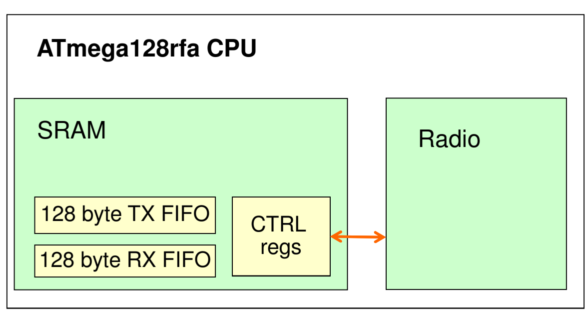


Power consumption? Range?
Data Rate? Cost?

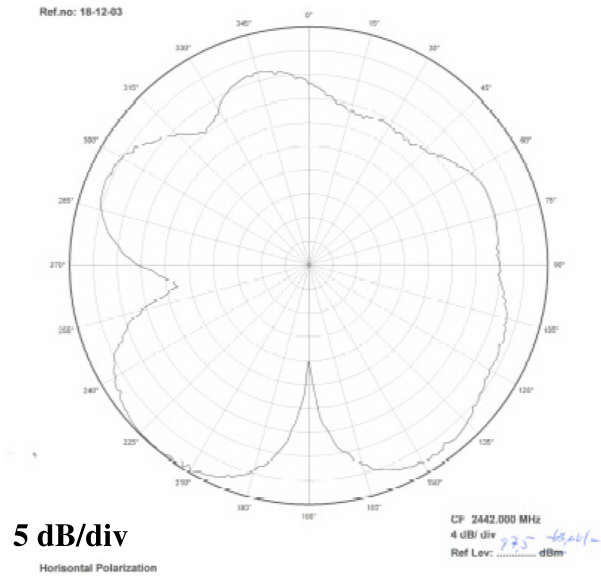
Communication Logistics

- Radio reads directly from SRAM
- 128-Byte Max Packet Size
- Built-in CRC (cyclic redundancy code)
- 128-bit AES encryption

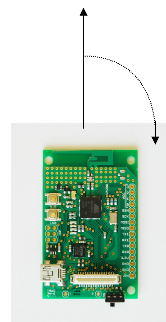




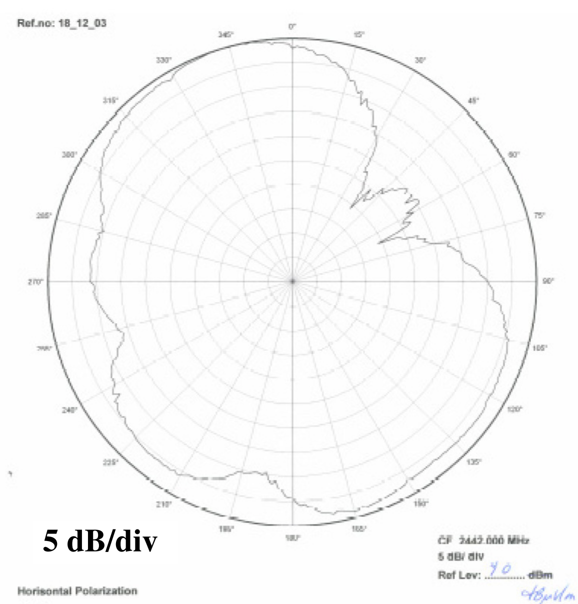
Horizontal Antenna Pattern



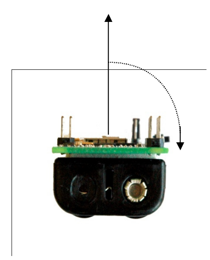
5 dB/div



Vertical Antenna Pattern

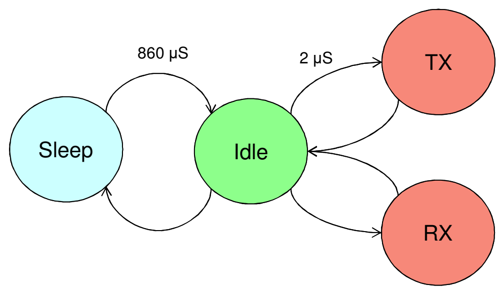


5 dB/div



CC2420 Power (similar to RF230)

- Idle: 426 µA
- Power Down: 20 µA
- Voltage Regulator Off: 0.2 µA
- TX: 17.4 mA
- RX: 18.8 mA



* All currents at 3 volts

Radio + CPU Power

2,870 mAH (@ 25ma) 1.5v x 2 (series)

- Active: 7 mA → 17 days
- Active + TX: 24.4 mA → 4.9 days
- Active + RX: 28.8 mA → 4.6 days
- Idle: 2 mA → 60 days
- Sleep: 70 µA → 4.68 yrs!

* Values based on FireFly v2.2

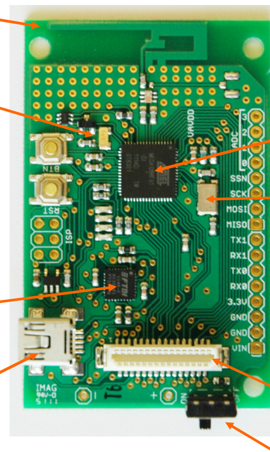
FireFly Board (1 of 3)

PCB Antenna

32.768 KHz OS Clock

Atmega128rfa1

16 MHz CPU / Radio Clock



USB->Serial from ftdi

Micro USB plug

Expansion Header

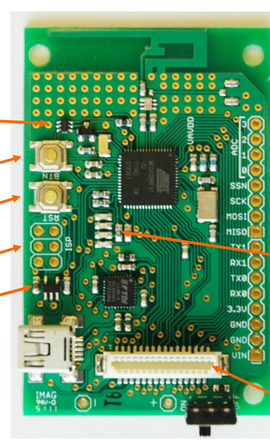
Power Switch

FireFly Board (2 of 3)

.1” GPIO Expansion

(4 ADC, 2 UARTs, SPI)

HW Clock Tuning



User Button

Reset Button

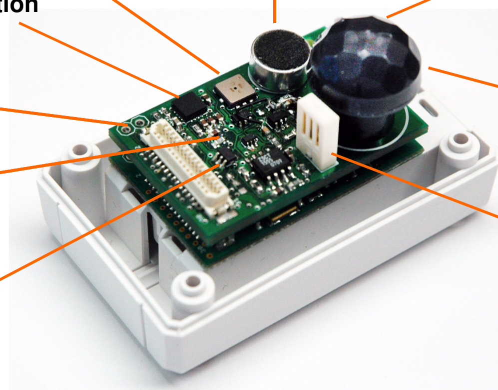
ISP Header

4 LEDs

Battery Input Voltage Regulator

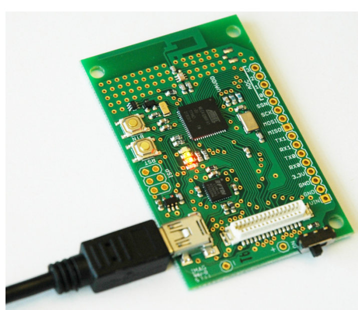
Expansion Header

FireFly Board (3 of 3)



FireFly Programming

- Nodes are programmed over a USB serial bootloader
- (programming is also possible via the ISP header)



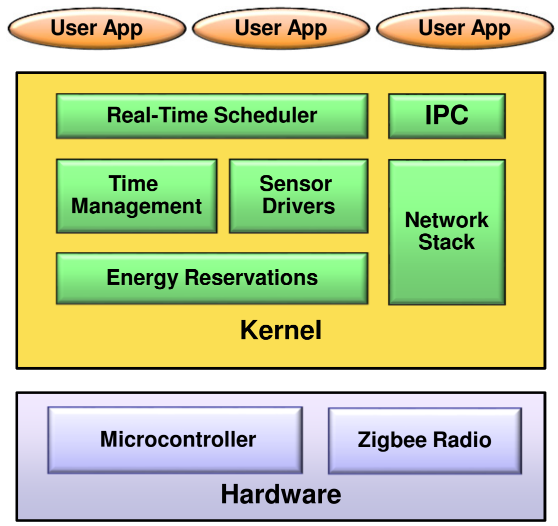
Software Tools

- AVR `gcc` 4.4.0
  - Compiler
- Binutils 2.17
  - `ld`, `as`, `objdump`, `size`, etc
- Avr-libc 1.4.6
  - `string.h`, `math.h`, `printf`, `atoi`, etc…
- `avrDUDE` 5.10
  - Downloading Utility
- Minicom (`TeraTerm`, `hyperterm`)
  - Serial Console

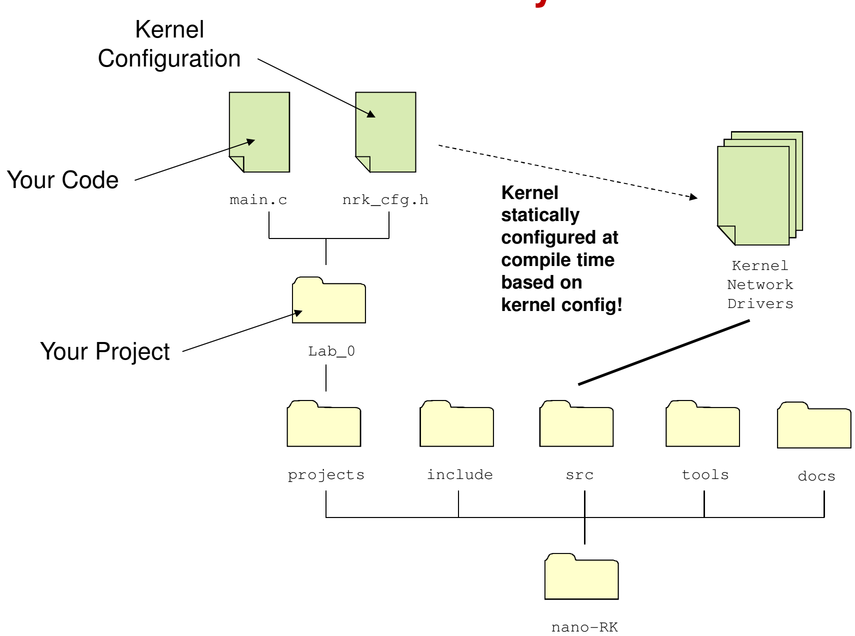
Nano-RK Operating System

- **C GNU tool-chain**
- **Classical Preemptive Operating System Multitasking Abstractions**
- **Real-Time Priority-Based Scheduling**
  - Rate Monotonic Scheduling
- **Built-in Fault Handling**
- **Energy-Efficient Scheduling based on *a priori* task-set knowledge (future lecture topic…)**

Nano-RK Architecture



Nano-RK Layout



Configure Nano-RK Task Set (nrk_cfg.h)

```
#define NRK_REPORT_ERRORS
// print error over serial

#define NRK_HALT_ON_ERROR
// stop the kernel if an error happens


// Enable Canary Stack Check
#define NRK_STACK_CHECK


// Max number of tasks in your application
// Be sure to include the idle task
// Making this the correct size will save on BSS memory which
// is both RAM and ROM...
#define NRK_MAX_TASKS                    5


#define NRK_TASK_IDLE_STK_SIZE         128
// Idle task stack size min=32
#define NRK_APP_STACKSIZE              128
#define NRK_KERNEL_STACKSIZE           128

#define NRK_MAX_RESOURCE_CNT           1
```


Nano-RK Reservations

- **CPU Utilization**
  - Time Allowed Per Period
  - For example, a task can run for 10ms of time every 250ms
- **Network Utilization**
  - Packets In and Out Per Period
- **Sensors & Actuators Usage**
  - Sensor Readings Per Period
- **{CPU, Network, Peripherals}**
  - Together comprise the total energy usage of the node
  - Static offline budget enforcement
  - Powerful tool for containment

Task Isolation

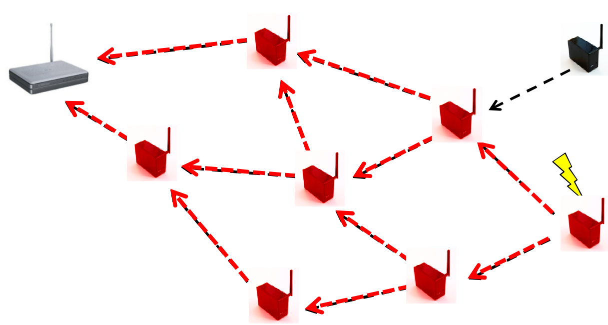
```
task1()
{

while(1)
   {

      do {
        // read sensor values
      } while(sensor_active()==1);

      // process sensor data

      // add to network queue

      nrk_wait_until_next_period();
   }


}
```

```
task2()
{

while(1)
   {

      // Check runtime data

      // Log to EEPROM

      nrk_wait_until_next_period();
   }

}
```

What if the sensor is broken and never finishes?

Network Isolation

- **Device failure can easily have global consequences**



Creating a Nano-RK Task

```
NRK_STK Stack1[NRK_APP_STACKSIZE];
nrk_task_type TaskOne;
void Task1(void);

...

nrk_task_set_entry_function( &TaskOne, Task1);
nrk_task_set_stk( &TaskOne, Stack1, NRK_APP_STACKSIZE);
TaskOne.prio = 1;
TaskOne.FirstActivation = TRUE;
TaskOne.Type = BASIC_TASK;
TaskOne.SchType = PREEMPTIVE;
TaskOne.period.secs = 0;
TaskOne.period.nano_secs = 250*NANOS_PER_MS;
TaskOne.cpu_reserve.secs = 0;
TaskOne.cpu_reserve.nano_secs = 50*NANOS_PER_MS;
TaskOne.offset.secs = 0;
TaskOne.offset.nano_secs= 0;
nrk_activate_task (&TaskOne);
```


Nano-RK Task Management

- **~1ms OS Tick Resolution**
  - **Variable Tick Timer (interrupts occur as required, not every quantum)**
- **`wait_until_xxx()` functions**
  - **Suspend task until the event or timeout happens**
  - **If there is no `wait_until_xxx()` call,**
    - **then your reserve will be violated**
    - **If reserves are disabled, then this can starve low-priority tasks and will waste battery power**

Typical Nano-RK Application

Task 1: Periodic

```
while(1)
{
// Do some great work
nrk_wait_until_next_period();
}
```

Task 2: Event-Based

```
while(1)
{
nrk_wait_until_rx_pkt();
// Check the packet
// Forward the packet
}
```

Task 3: Absolute Wait

```
nrk_time_t t;
t.secs=5;
t.nano_secs=0;
while(1)
{
nrk_time_get(&t);
// Do some work
t.secs +=5;
nrk_wait_until( t );
}
```

Task 4: Relative Wait

```
nrk_time_t t;
t.secs=5;
t.nano_secs=0;
while(1)
{
// Do some work
nrk_wait( t );
}
```

Kernel automatically goes into sleep mode when no task is executing.

Lets the Kernel save power by not looping forever. This would cause a CPU reserve violation anyway.

Nano-RK Fault Handling

- **Task Time Violations**
  - **OS will enforce time bounds given to a task**
- **Canary Stack Check**
  - **Check if user-specified stack has been overflowed**
  - **Not 100%, but incurs low overhead and better than nothing**

- **Unexpected Restarts**
  - **Capture restart that occurs without power-down**
- **Resource Over-use**
  - **Manage sensors / actuators**
- **Low-Voltage Detection**
- **Watchdog Timer**

Sample Nano-RK Task

```
void Task1()
{
  uint16_t cnt, buf;
  int8_t fd, val;

  printf( "My node's address is %d\r\n", NODE_ADDR );
  printf( "Task1 PID = %d\r\n", nrk_get_pid());

  // Open ADC device as read
  fd = nrk_open(FIREFLY_SENSOR_BASIC, READ);
  if (fd == NRK_ERROR)
     nrk_kprintf(PSTR("Failed to open sensor driver\r\n"));

  cnt = 0;
  while (1) {
    nrk_led_toggle(BLUE_LED);
    // Example of setting a sensor
    val = nrk_set_status(fd, SENSOR_SELECT, BAT);
    val = nrk_read(fd, &buf, 2);
    printf( "Task1 bat=%d", buf);
    val = nrk_set_status(fd, SENSOR_SELECT, LIGHT);
    val = nrk_read(fd, &buf, 2);
    printf( " light = %d", buf);
    cnt++;
  }
    nrk_close(fd);
}
```


Nano-RK Network Link Layers

- **LPL-CSMA (BMAC)**
  - Low-Power Listen Carrier Sense Multiple Access
  - Contention-based Protocol
  - Polastre, Hill, Culler, "*Versatile Low Power Media Access for Wireless Sensor Networks*", Sensys 2004
- **RT-Link**
  - Time-Division Multiple Access (TDMA) Protocol
  - Anthony Rowe, Rahul Mangharam, Raj Rajkumar, "*RT-Link: A Time-Synchronized Link Protocol for Energy Constrained Multi-hop Wireless Networks*," SECON 2006

Receiving a Packet in Nano-RK

```
void rx_task()
{
  uint8_t i,len;
  int8_t rssi,val;
  uint8_t *local_rx_buf;

  // init bmac on channel 25
  bmac_init(25);
  bmac_rx_pkt_set_buffer(rx_buf,RF_MAX_PAYLOAD_SIZE);

  while (1) {
        // Wait until an RX packet is received
        val = bmac_wait_until_rx_pkt();

        // Get the RX packet
        local_rx_buf = bmac_rx_pkt_get(&len, &rssi);
        printf( "Got RX packet len=%d RSSI=%d [", len, rssi );
        for (i =0; i < len; i++ )
            printf( "%c", local_rx_buf[i]);
        printf( "]\r\n" );

        // Release the RX buffer so future packets can arrive
        bmac_rx_pkt_release();
  }

}
```


Sending a Packet in Nano-RK

```
void tx_task()
{
  uint8_t j, i,val,len,cnt;

  printf( "tx_task PID=%d\r\n",nrk_get_pid());

  // Wait until the tx_task starts up bmac
  // This should be called by all tasks using bmac that
  // do not call bmac_init()...
  while (!bmac_started()) nrk_wait_until_next_period();
  cnt = 0;
  while (1) {
        // Build a TX packet
        sprintf( tx_buf, "This is a test %d",cnt );
        cnt++;

        // Transmit the packet
        val = bmac_tx_packet(tx_buf, strlen(tx_buf));

        // Task gets control again after TX complete
        nrk_kprintf( PSTR("TX task sent data!\r\n") );
        nrk_wait_until_next_period();
   }

}
```

Tips and Tricks (1 of 4)

- **Don't Allocate Large Data Structures Inside Functions**
  - Allocating large data structures in functions puts them on the stack
  - Make them global if need be (bad style for a PC, but this isn't a PC)
  - Stack is usually 128 bytes!
- **Take Care When Passing Large Data Types to Functions**
  - Pass large structures by reference using pointers so less data gets pushed on the precious stack.
- **Avoid Recursive Function Calls**
  - Recursive function calls keep pushing onto the stack each time they recurse
- **Use "`inline`" For Speed And To Save Stack Space**
  - "`inline`" in C avoids function calls and (you guessed it) doesn't push onto the stack

Tips and Tricks (2 of 4)

- **Be very careful with Dynamic Memory**
  - `malloc` does work, but can cause fragmentation and all sorts of other problems. Use with EXTREME care or better yet *not at all*.
- **Watch out for strings**
  - Strings declared anywhere consume DATA and hence use RAM.
  - They don't show up using `avr-nm` (listing of symbols)
  - Sometimes it is better to pass a numerical value to a function that has a big `kprintf()` switch inside it.
- **Use `nrk_kprintf()` whenever possible for constant strings**
  - `nrk_kprintf()` stores strings in flash memory using the `PSTR()` macro.
  - Only use regular `printf()` when the string is dynamic (i.e. you use `%d` to print variables etc).

Tips and Tricks (3 of 4)

- **How Much Memory Is My Code Using?**
  - .data is the amount of RAM that your program uses that is defined at startup as a particular value.
    - Consumes RAM and ROM
  - .bss is the amount of zeroed-out RAM your program uses.
    - Consumes RAM only
  - **RAM = .data + .bss (+ Kernel Stack)**
  - **Flash = .data + .text**
  - Stack appears in .bss section EXCEPT for Kernel, so add Kernel stack to RAM figure

```
Size after:
main.elf  :
section      size      addr
.data         220   8388864
.text       17258         0
.bss         1021   8389084
.stab       41268         0
.stabstr    16934         0
Total       76701
```

RAM = 220+1021+128 = 1,369 bytes

Flash = 17258+220 = 17,478 bytes

Total RAM = 8,192 bytes

Total ROM = 131,072 bytes

Tips and Tricks (4 of 4)

- **What variables are using up my memory?**
  - Use `avr-nm` (NAME) to find a list of symbols and how much they consume. Below is an example that prints the size of functions and static memory sorted as decimal:

```
avr-nm -S --radix=d --size-sort main.elf
...
(address)   (size)
08388989 00000001 D NRK_UART1_TXD
...
08389446 00000116 B tx_buf
00012074 00000118 T nrk_event_wait
```

- "T" refers to the text section, "B" refers to the BSS section, and "D" refers to the data section. Remember that strings do not show up in this list because they do not have compiler-mapped labels.

Nano-RK Resources

- Nano-RK Website
  - http://www.nano-rk.org/nano-RK
- Kernel Documentation
  - http://www.nanork.org/nano-RK/wiki/nrk-api
- Link Layer Documentation
  - http://www.nanork.org/nano-RK/wiki/bmac-api
  - http://www.nanork.org/nano-RK/wiki/RT-Link
- Sensor Driver Documentation
  - http://www.nanork.org/nano-RK/wiki/firefly-basic-sensor-driver
- Example Projects
  - http://www.nanork.org/nano-RK/browser/nano-RK/projects

Questions?

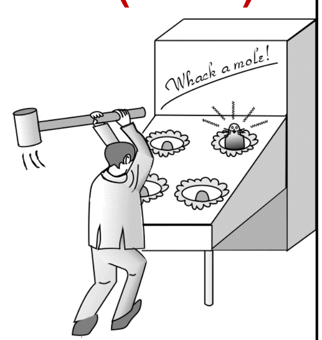
Lab #0: “Hello World!”

- **Goals**
  - Get to know your Hardware
  - Get to know the Tool-Chain
    - Compile and Download “Hello World” Program
  - Get to know your Wireless Radio
    - Measure RSSI* and Packet Loss
    - Estimate your Path-Loss exponent
  - Profile code execution time
    - Investigate the Scheduler

"Step one: take a good stiff drink."

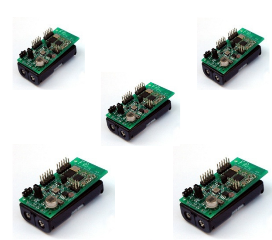
RSSI: *Received Signal Strength* Indicator, measures received radio signal strength (energy integral).

Lab #1: Distributed Whack-a-Mole (1 of 2)



Rules of the game

- LEDs on *one* node at a time should illuminate and sequentially count down
- Blocking the light sensor should effectively “whack the mole” and initiates a new faster count down on a different random node
- If all LEDs turn off on a node before being whacked, then the game halts printing a score based on how many moles you hit as well as your completion time
- If all nodes are caught on time, the game completes, victoriously flashes many LEDs and prints the time taken to whack all moles
- An interactive terminal program should allow the configuration and start of new games



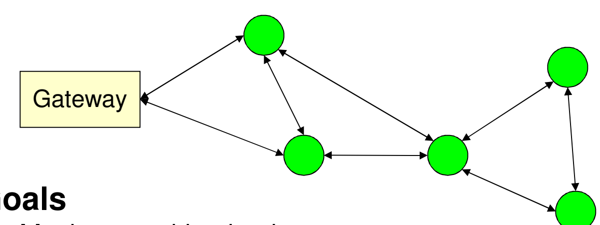
Lab #1: Distributed Whack-a-Mole (2 of 2)

Goals

- Read and Process Basic Sensor Values
  - We do not want to see false positives or false negatives!
- Basic Two-Way Serial Interaction
- Communication and Coordination Between Multiple Nodes
  - Various Possible Architectures (keep it simple)
  - Single-Hop Communication is fine for now
- Develop and Debug in a Distributed Resource-Constrained Environment

Lab #2: Multi-Hop Communications

- **Task**
  - Build a Self-Configuring Sensor Network
  - Automatically Capture a 5-node network topology
  - Send Sensor Data across multiple hops to a gateway
  - Design Your Own Network Layer



- **Goals**
  - Mesh networking basics
  - Gets You Ready for your Projects!

Summary

- **FireFly Hardware**
  - Atmel processor features
  - Chipcon Radio
  - FireFly Node
  - FireFly Programmer
- **Nano-RK Operating System**
  - Configuration
  - Tips and Tricks
- **Lab Descriptions**