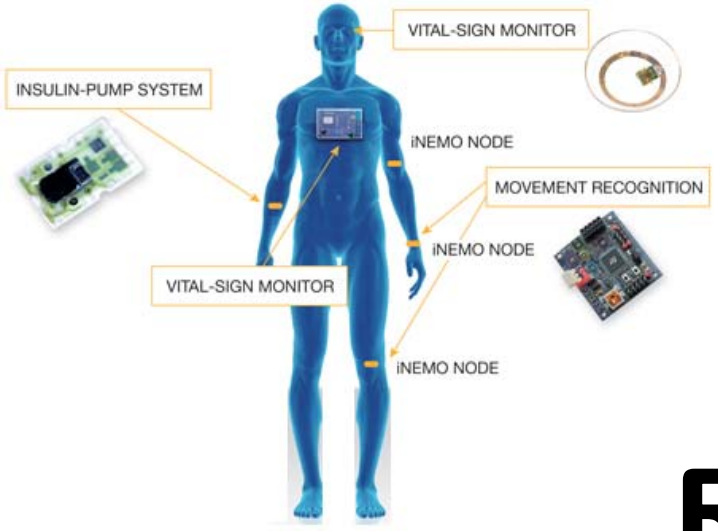
Embedded Software Architecture



Real Time Embedded Systems

www.atomicrhubarb.com/embedded

Lecture 1 – January 17, 2012

Topic

Section Topic

- Where in the books
  - Catsoulis chapter/page
  - Simon chapter/page
  - Zilog UM197 (ZNEO Z16F Series Flash Microcontroller Contest Kit User Manual)
  - Zilog UM171 (ZiLOG Developer Studio II—ZNEO User Manual)
  - Zilog PS220 (ZNEO Z16F Series Product Specification)
  - Zilog UM188 (ZNEO CPU Core User Manual)
  - Assorted datasheets

Survey Of Embedded Software Architectures

- Round Robin
- State Machine
- Round Robin with Interrupts
- Just interrupts
- Function Queue Scheduling
- Real-Time Operating System

Round Robin

- Round Robin / Control Loop
- Everything is a function call from the main loop

```
main {
    ...
    while(1) {
        check_buttons();
        scan_display();
        read_tempsensor();
        operate_motor();
    }
}
```

Round Robin

- Low priority tasks need to be slowed down

```
while(1) {
        ...
            if (display_skips<1000) {
                display_skips++;
            }
            else {
                NUM1++;
                display_skips=0;
            }
            LEDisplay_hex(NUM1);
        ...
    }
```

Round Robin

- Priority – None, everything runs in sequence.
- Response time – The sum of all tasks.
- Impact of changes – Significant. Changing the execution time of tasks or adding tasks impacts all other tasks.
- Simplicity, no shared data problems.

State Machine

```
while(1) {
   switch(state) {

       case IDLE:
         check_buttons();
         LEDisplay_hex(NUM1);
         if (BUTTON1 | BUTTON2 | BUTTON3)
             state=SHOW;
         break;

       case SHOW:
           NUM1=0;
           if (BUTTON1) NUM1 += 0x0001;
           if (BUTTON2) NUM1 += 0x0010;
           if (BUTTON3) NUM1 += 0x0100;
           state=IDLE;
           break;
   }
}
```

State Machine

- Similar to round robin, but only the current state gets executed.

Round Robin with Interrupts

```
SET_VECTOR(P3AD,button_isr);
SET_VECTOR(TIMER1, display_isr);

while(1) {

        read_temp();

}
```

Round Robin with Interrupts

- Priority – Interrupts get priority over main loop
  - Priority of interrupts as well
- Response time –
  - The sum of all tasks or
  - Interrupt execution time
- Impact of changes – Less significant for interrupt service routines. Same as Round Robin as main loop.
- Shared data – must deal with data shared with interrupt service routines

Just interrupts

```
SET_VECTOR(P3AD,button_isr);
SET_VECTOR(TIMER1, display_isr);

while(1) {
    ;
}
```

Just interrupts

- Can have problems if too many ISRs
- If a high priority interrupt takes longer to execute than lower priority interrupts, then some will get missed.
  - Or you need to deal with nested interrupts (Zilog AN0141 describes how to do this).

Function Queue Scheduling

- Function pointers are added to a queue.
- The main loop cycles through the queue and executes tasks.
- Tasks or interrupts add new tasks to the function queue.

Function Queue Scheduling

```
#define MAX_TASKS 20
typedef int(*FuncPtr);
FuncPtr tasks[MAX_TASKS]
int current_task = 0;

void add_task(FuncPtr func) {
    int n;
    for(n=current_task+1;n<MAX_TASKS-1;n++) {
        if(tasks[n]==NULL) {
            tasks[n]=func;
            return;
        }
    }
    for(n=0;n<current_task;n++) {
        if(tasks[n]==NULL) {
            tasks[n]=func;
            return;
    }
}
```

```
void display_task() {
    LEDisplay_hex(NUM1);
    add_task(button_task);
}

void button_task() {
    check_buttons();

    NUM1=0;
    if (BUTTON1) NUM1 += 0x0001;
    if (BUTTON2) NUM1 += 0x0010;
    if (BUTTON3) NUM1 += 0x0100;

    add_task(display_task);

}
```

```
main() {

       LEDisplay_init();
    LEDisplay_clear();
    init_buttons();

    add_task(button_task);

      while(1) {
        if(tasks[current_task]==NULL) {
            ;
        }
        else {
            (*tasks[current_task])();
            tasks[current_task]=NULL;
        }
        current_task++;
        if(current_task>=MAX_TASKS) current_task=0;
    }
}
```

Function Queue Scheduling

- Priority – Interrupts have priority. Tasks execute in sequence
- Response time – Execution time of the longest task
- Impact of changes – Low. Interrupts manage priority functions. Queue manages lower priority.
- Shared data – must deal with data shared with interrupt service routines

Function Queue Improvements

- Include time scheduling

```
typedef int(*FuncPtr);

typedef struct {
    long timer;
    int status;
    FuncPtr;
} Task;

Task task_list[MAX_TASKS];
```

An interrupt decrements all task timers. When it reaches 0 its available for execution

Function Queue Improvements

- Include task priority

```
typedef int(*FuncPtr);

typedef struct {
    int priority;
    FuncPtr;
} Task;

Task task_list[MAX_TASKS];
```

Highest priority tasks get moved to the head of the queue.

Function Pointers

- The Function Pointer Tutorial
  - http://www.newty.de/zip/e_fpt.pdf

Real-Time Operating System

- The RTOS switches between several tasks. Can suspend one task to complete another.
- Allows us to higher priority tasks first, or give the appearance of several tasks executing simultaneously.
- System response time can be relatively stable.
- Switching tasks (context switching) requires overhead

Z8 RTOS

- Several RTOSs available for Z8 Encore
  - CMX Real-Time Software for the Z8 Encore
    - http://www.cmx.com/zilog/
  - ECROS Operating System
    - $5 per CPU. $300 source license.
    - http://www.ecrostech.com/Products/Z8Encore/Ecros/Intro.htm
  - ZRT - A Real-Time Operating System for the Z8 Microcontroller
    - free source
    - http://www.jandspromotions.com/zilog2003/m-4172.htm
    - Article in Circuit Cellar, June 2004

Z16 RTOS

- Several RTOSs available for Z16
  - CMX Real-Time Software for the Z16
  - Nuttx RTOS - A Real-Time Operating System for a number of small microcontrollers
  - open source
  - http://www.nuttx.org

Real-Time Operating System

```
void countUp(void)
{
	for (;;) {
		delay(0x40);
		upcounter++;
		loadLEDData(upcounter, 2);
	}
}

void countDown(void)
{
	for (;;) {
		delay(0x40);
		downcounter--;
		loadLEDData(downcounter, 3);
	}
}
```

Real-Time Operating System

```
main ()
{
      initLED();
      initThreads();

      createThread(countUp);
      createThread(countDown);
      createThread(displayData);

      EI();
      return;
}
```

Init RTOS

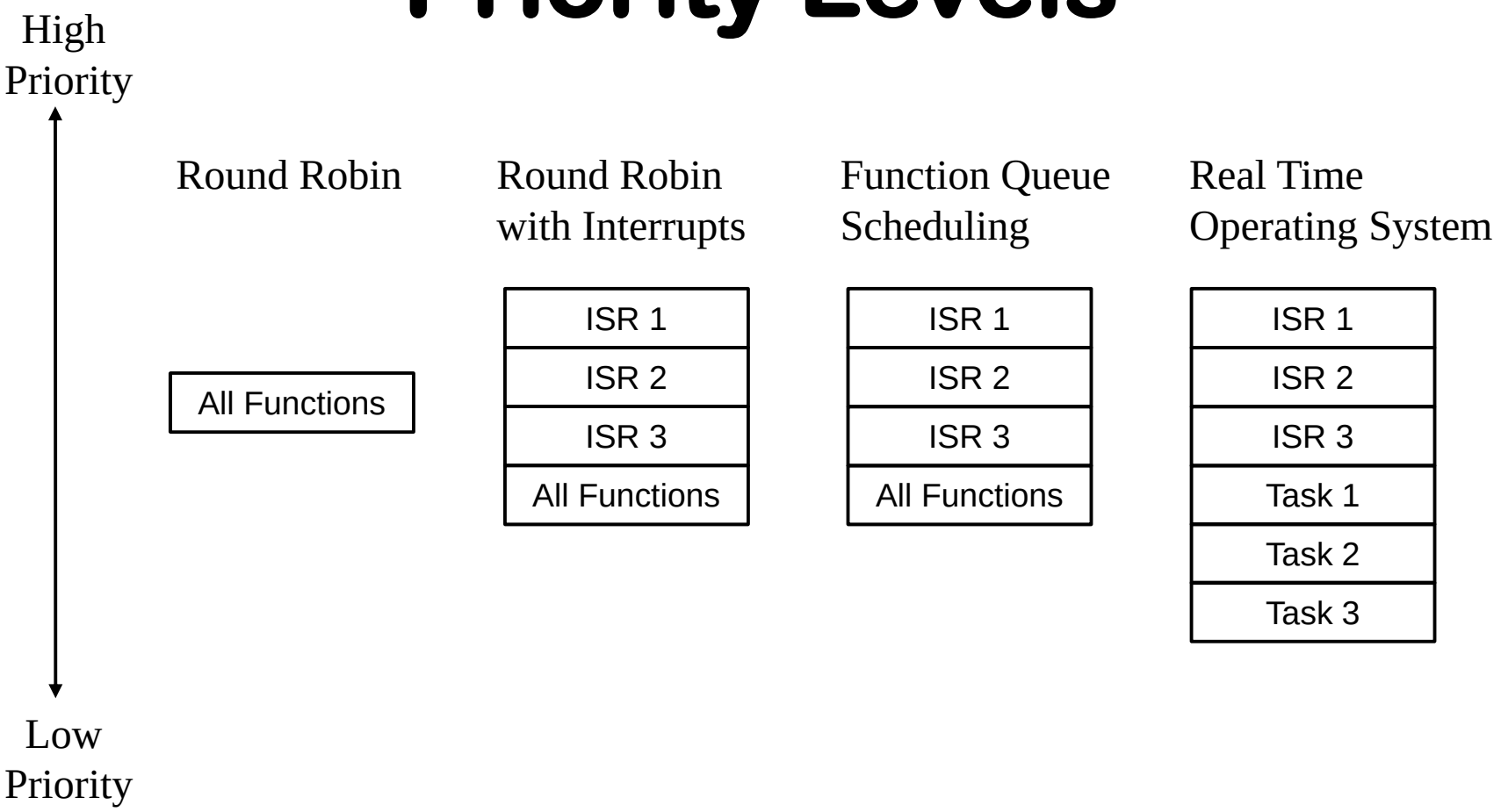
Create Threads

No loop!

RTOS

- Priority – Interrupts have priority. Tasks execute in priority order.
- Response time – Can be as low as zero (plus execution time for interrupts).
- Impact of changes – Low.
- Shared data – Must deal with data shared with interrupt service routines.

Priority Levels



Summary

- Some extensions to standard C (but common to embedded systems) to allow for special concerns: interrupts, memory, optimization.
- Several common software architectures suitable for different embedded systems requirements.

Why ... ?

End of Section Reminder

References