Lecture 10: Scheduling II, Scheduling worksheets

Vikram Iyer

Adapted from material by Blake Hannaford

29 9:30-11:30 OH (Deeksha) ECE 345 11:30-13:30 OH (Alex) ECE 345	30 13:00-15:00 OH (Zach) ECE 345	01 9:30-11:30 OH (Deeksha) ECE 345 12:30-14:20 Lecture MOR 230 *Lecture 10: Scheduling II* 14:00-15:00 OH (Vikram) ECE 345	02 10:00-12:00 OH (Zach) ECE 345 12:30-14:30 OH (Alex) ECE 345	03 12:30-14:20 Lecture MOR 230 *Lecture 11: Scheduling III and Lab 3 Intro* 14:00-15:00 OH (Vikram) ECE 345

May

Monday	Tuesday	Wednesday	Thursday	Friday
06 9:30-11:30 OH (Deeksha) ECE 345 11:30-13:30 OH (Alex) ECE 345 23:59 Lab 2 due	07 13:00-15:00 OH (Zach) ECE 345	08 9:30-11:30 OH (Deeksha) ECE 345 12:30-14:20 Lecture MOR 230 *Lecture 12: Midterm Review* 14:00-15:00 OH (Vikram) ECE 345	09 10:00-12:00 OH (Zach) ECE 345 12:30-14:30 OH (Alex) ECE 345	10 12:30-14:00 Midterm
13 9:30-11:30 OH (Deeksha) ECE 345 11:30-13:30 OH (Alex) ECE 345	14 13:00-15:00 OH (Zach) ECE 345	15 9:30-11:30 OH (Deeksha) ECE 345 12:30-14:20 Lecture MOR 230 *Lecture 13: I2C and SPI Communication* 14:00-15:00 OH (Vikram) ECE 345	16 10:00-12:00 OH (Zach) ECE 345 12:30-14:30 OH (Alex) ECE 345	17 12:30-14:20 Lecture MOR 230 *Lecture 14: UART Serial Communication* 14:00-15:00 OH (Vikram) ECE 345
20 9:30-11:30 OH (Deeksha) ECE 345 11:30-13:30 OH (Alex) ECE 345 23:59 Lab 3 due	21 13:00-15:00 OH (Zach) ECE 345	22 9:30-11:30 OH (Deeksha) ECE 345 12:30-14:20 Lecture MOR 230 *Lecture 15: Intro to FreeRTOS* 14:00-15:00 OH (Vikram) ECE 345	23 10:00-12:00 OH (Zach) ECE 345 12:30-14:30 OH (Alex) ECE 345	24 12:30-14:20 Lecture MOR 230 *Lecture 16: Intro to Critical Sections and Semaphores* 14:00-15:00 OH (Vikram) ECE 345

Announcement/Reminders

- Lab 2 updated spec + due date: due Monday 5/6
  - Tip: Start by just trying to get the timer to output to a pin, and then adjust the timing. Might be easier to debug once you have an output
- Midterm on 5/12
  - During class here 90 min
  - Goal: make sure you know concepts, labs are worth more

Last time and plan for today

Last time:

- Reading analog values
- Intro to interrupts

Plan for today:

- Timer help
- Continue with more advanced scheduling concepts
  - How to build a scheduler
  - Halting tasks, keeping track of functions
  - Worksheet examples

Timer setup example + misconceptions #99

Vikram Iyer STAFF
11 minutes ago in Labs

PIN STAR WATCHING 1 VIEW

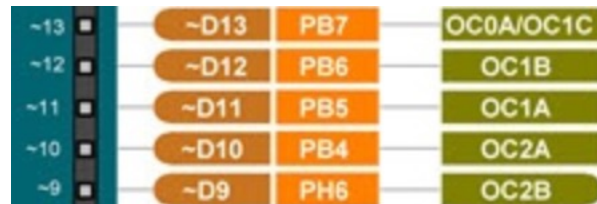
Here's a block of code that explains in detail how to set up a timer. Note that I've written this code with some macros that you'll have to define based on tables in the datasheet, but hopefully this helps you get started and shows the key registers you need to set along with the tables to refer to in the datasheet. I'll go over this in lecture tomorrow as well.

```
void setup() {
  /* Step 1: Pick a pin and set it up
  as an output. Use pin diagram to figure out
  which pin maps to which timer output and
  rewrite this line by changing the appropriate DDR
  register from part 1 of the lab. */

  pinMode(11, OUTPUT); // REPLACE WITH YOUR CODE

  /* Step 2: Clear Timer1 control registers
  Set all the bits to 0 to remove old settings
  and put it in a known state. TCCR1A/B are the
  control registers for Timer1. These macros are
  predefined so you don't have to look up a
  specific register address. */
  TCCR1A = 0; // Timer/Counter Control Register A
```

Common issues

- Registers vs bitmasks: Registers (TCCRnB, OCRnA, etc.) are not the same as bitmasks (WGM, COM, CS, etc). A register is a value that you can set that enables functionality on the chip. The bitmasks are specific bit positions in the registers. The bitmask macros are defined to make it easy to set them.

- Arduino pins have multiple names and functions. For example, pin 11 (label on the side of your Arduino) refers to pin Port B5 (PB5) and is also connected to Timer1 output OC1A.



- Read the data sheet to understand the timer modes and bits to set. Most of the content you need is on pages 145-147, and 154-155.

- The timer is a dedicated hardware block that keeps running in the background. You don't need anything in your loop function other than to manipulate it (turn it on/off, change the sound).

Scheduler types

1. Non-preemptive Round Robin (RR)- Basic `while(1)` loop
2. Synchronized non-preemptive RR- The same loop with a timing function for synchronization
3. Modifications to Synchronized RR to enable different task periods
4. Adding a `sleep(int d)` delay option for tasks
5. Making the scheduler dynamic with a `halt_me()` function

333µs
666µs
1 ms

Scheduler		Task A	Task B	Task C

```
while(1) {
   taskA();
   taskB();
   taskC();
}
```

Task	P	C	D
A	1	0.2	P
B	2	0.2	P
C	2	0.2	P

ms

How to Build a Scheduler

So far we've had very simple programs

```
while(1) {
  taskA();
  taskB();
  taskC();
  time_delay();
}
```

How to Build a Scheduler

New features we want to add

1. Ability to keep track of a function’s state and start it
2. Ability for a task to halt itself
3. Ability of a task to sleep and let other tasks run

Part I: Using function pointers to start a task

We learned that the syntax:

```
type (*name)(type arg1, type arg2, ...)
```

indicates a function pointer called name which can point to any function having the same prototype. For example

```
int (*fp)(char x, int y, double z)
```

```
void start_function(void (*functionPTR)() ){
  functionPTR();
}
```

Part I: Using function pointers to start a task

Now our basic scheduler could be written:

```
while(1) {
  start_function( taskA);
  start_function( taskB);
  start_function( taskC);
  time_delay();
}
```

function ptr

This version is a trivial change but this is an important step

Part II: Keeping lists of functions

We can create an array of function pointers as follows:

```
#define NUMBER_OF_FUNCTIONS 10

void (*list_of_functions[NUMBER_OF_FUNCTIONS])(void *p)
```

Example:

```
#define NT_RUNNING 10
void (*runningTasks[NT_RUNNING])(void *p)
```

... a list of all the tasks which are currently running.

Part II: Keeping lists of functions

We can create an array of function pointers as follows:

```
#define NUMBER_OF_FUNCTIONS 10

void (*list_of_functions[NUMBER_OF_FUNCTIONS])(void *p)
```

Example:

```
#define NT_RUNNING 10
void (*runningTasks[NT_RUNNING])(void *p)
```

... a list of all the tasks which are currently running.

We can use a NULL pointer at the end of the list if there are less than NT RUNNING active tasks (similar to the zero byte at the end of a string).

```
//function prototypes for tasks
void taskA(void *p);
void taskB(void *p);
...

// function prototype for
scheduler void scheduler();


#define NTASKS 4

// lets make the task list global
// an array of function pointers,
// each one has a void*
// parameter.

void (*readytasks[NTASKS])
  (void *p);
```

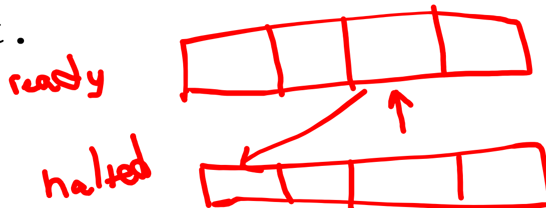
```
//continued...
main() {
  // initialize array of pointers
  // to tasks

  readytasks[0] = taskA;
  readytasks[1] = taskB;
  ...

  // NULL signals the last task
  readytasks[2]= NULL;

  // now start scheduler
  while(1) {
    scheduler();
    time_delay(); // 1 ms timer
  }
} // end of main
```

Scheduler function (pseudocode)

```
scheduler() {
  if(readytasks[task_index] == NULL && task_index != 0) {
    task_index=0;
  }
  if(readytasks[task_index] == NULL && task_index == 0){
    // figure out something to do because
    // there are no tasks to run!
  } else {
    start_function(readytasks[task_index]);
    task_index++;
  }
  //   Round Robin/we're taking turns
  return;
}
```

Part III: Manipulating the Task Lists

```
halt_me() {
  // 1. Identify which task is currently running (i.e.
  //    look at task_index)

  // 2. Copy the function pointer from
  //    readytasks[task_index] to the
  //    haltedtasks array

  // 3. Move the remaining tasks up in readytasks[] to
  //    fill the empty hole and copy NULL into the
  //    last element.

  return;
}
```



Part III: Manipulating the Task Lists

Watch out for potential logic bug

```
if (time_to_halt) {
  halt_me();


}
```

Part III: Manipulating the Task Lists

Watch out for potential logic bug

```
if (time_to_halt) {
  halt_me();
  return;  // Need to make sure the function returns
}
```

Need to make sure the code actually returns after calling `halt_me()`

Typical Task using sleep()

```
task_c(p) {
  do_computation(); // compute for a little while
  if(time_to_sleep){
    sleep(10);
    return;
  }
  else {
    do_computation(); // compute for a little while
  }
  return;
}
```

Sleep function (pseudocode)

```
sleep(int d) {
  // 1. Copy function pointer from
  //    readytasks[task_index] to the
  //    waitingtasks[] array.
  // 2. Clean up readytasks[] as in halt_me();
  //    copy d into the delays array with the same
  //    index as the function pointer has in
  //    waitingtasks[]
}
```

Keeping track of sleep delays

```
  // continued...
  // 5. for each element of waitingtasks which is
  //    not NULL: decrement the delay value d.
  //    if (d==0) move the function pointer to the
  //    end of the readytasks[] array and remove
  //    it from waitingtasks.
  // end of scheduler
  return;
}
```

Alternative Data Structure for Scheduling

Let's set up a struct which contains everything we need to track about a Task:

```
typedef struct TCBstruct {
  void (*ftpr)(void *p);     // the function pointer
  void *arg_ptr;             // the argument pointer
  unsigned short int state;  // the task state
  unsigned int delay;        // the task state
} ;
```

```
#define STATE_RUNNING  0
#define STATE_READY    1
#define STATE_WAITING  2
#define STATE_INACTIVE 3
TCBStruct TaskList[N_MAX_TASKS];

// Then we can set up the task // list as follows:
int j=0;
int task_B_Arg;

TaskList[j].ftpr = task_A();
TaskList[j].arg_ptr = NULL;
TaskList[j].state = STATE_INACTIVE;
TaskList[j].delay = 0;
j++;
```

```
// continued… start task B
TaskList[j].ftpr = task_B();

// (example: let's say we need an arg value of 56)
task_B_Arg = 56;
int *ip = &task_B_Arg;
TaskList[j].arg_ptr = (void*)ip;
TaskList[j].state = STATE_READY;
TaskList[j].delay = 0;
j++;

TaskList[j].fptr = NULL;        // marks end of list

... maybe other tasks   ...
```

Examples of halt_me(), start_task() and delay()

```
halt_me() {

}

start_task(int task_id) {

}

delay(int d) {


}
```

Examples of halt_me(), start_task() and delay()

```
halt_me() {
  TaskList[t_curr].state = STATE_INACTIVE;
}

start_task(int task_id) {

}

delay(int d) {


}
```

Examples of halt_me(), start_task() and delay()

```
halt_me() {
  TaskList[t_curr].state = STATE_INACTIVE;
}

start_task(int task_id) {
  TaskList[task_id].state =  STATE_READY;
}

delay(int d) {



}
```

Examples of halt_me(), start_task() and delay()

```
halt_me() {
  TaskList[t_curr].state = STATE_INACTIVE;
}

start_task(int task_id) {
  TaskList[task_id].state =  STATE_READY;
}

delay(int d) {
  TaskList[t_curr].delay = d;
  TaskList[t_curr].state = STATE_WAITING;
}
```