Schedule and Syllabus

- Lecture
  - Linux Device Drivers
  - Linux IPC and Synchronization
  - Embedded Networking
  - Distributed Control
  - Safety
- Lab
  - Interrupt Driven Device Driver
  - Project of your Choice
- Quiz (11/13)
  - Timing analysis (Timing Diagrams)
  - RTOS Basics (Latency, Signal, Wait, Multi-Threading, Resources)
  - Process/Thread Synchronization
  - Device Drivers in Linux
  - Synchronization and IPC in Linux

Lab Discussion

- The Sonar: Where you need capacitors
- The Intrinsyc Board and Breakout Board Overview
  - Hardware
  - Boot Sequence and Options
- Power Supply Setup w/ Sonar
- Programmed I/O from Kernel Space (In the Device Driver)
  - GPDR
  - GPSR
  - GPCR
  - GPLR
- Use Port0 to Read (GPIO 5 or 6 works for sure) (INIT)
- Use Port1 to Write (GPIO 11,12,13 work for sure) (ECHO)

Reference

- Cerfboard HW spec
- device driver skeleton

Driver Template

```
#define SONAR_DEV_MAJOR      23              /* arbitary choice for major number */
#define SONAR_IDENT "Sonar Device Driver"
#define SONAR "Sonar"

/* Useful GPIO macros are defined in the linux/include/asm-arm/arch/SA-1100.h */
/* --- function declarations --- */
….
/* The file operation struct. You should not need to change this */
static struct file_operations hello_file_ops = {NULL, NULL, Sonar_read, Sonar_write,
      NULL,NULL,NULL,NULL,Sonar_open,NULL,hello_release,NULL,NULL };


int Sonar_init(void) {  // gets called during system boot
  printk("\n\nSonar_init called\n\n");
  if (lock==true) {
    printk("Device Sonar already installed .\n");
    return -1; };
  lock=true;
  if (register_chrdev(Sonar_DEV_MAJOR,Sonar,&Sonar_file_ops)) {
    printk("%s, cannot register to major device %d\n",
         Sonar_IDENT,Sonar_DEV_MAJOR);
    return 1;
  }
  else {
    printk("Installed %s\n",Sonar_IDENT);
    return 0;
  }
}
```

Driver Template

```
int Sonar_open(struct inode *inode,struct file *file) {
  /* Grab any resources our driver needs, including interrupt lines, memory ranges, ports */
  /* register interrupt routines */
 /*  manage mutual exclusion if necessary */
  return 0;
}

int Sonar_release(struct inode *inode,struct file *file) {
  /* Release any resources our driver has */
  return 0;
}

static ssize_t Sonar_read(struct file * file, char * buffer, size_t count,
                          loff_t *ppos) { // be sure to honor blocking/nonblocking settings from open
  /* Use the function below to copy from kernel to user space */
  // __copy_to_user((void*)buffer,(void *)str,count))
  return 0;
}

static ssize_t Sonar_write(struct file * file, const char * buffer,
                           size_t count, loff_t *ppos) { // be sure to honor blocking/nonblocking settings
  /* Use the function below to copy from user to kernel space */
  // __copy_from_user((void*)str,(void *)buffer,count))
  return 0;
}
```

Back to Sonar Driver

- Simple case

Sonar w/ Better Properties