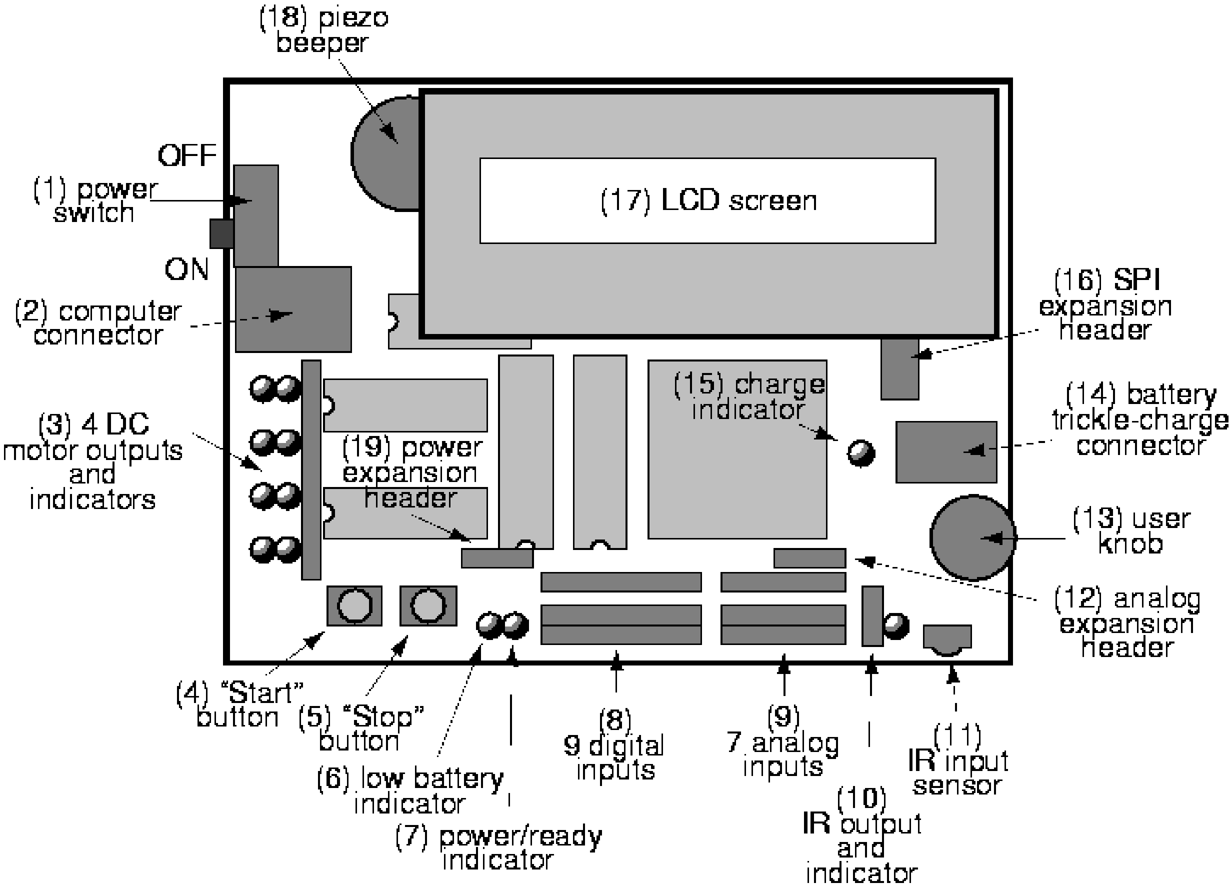
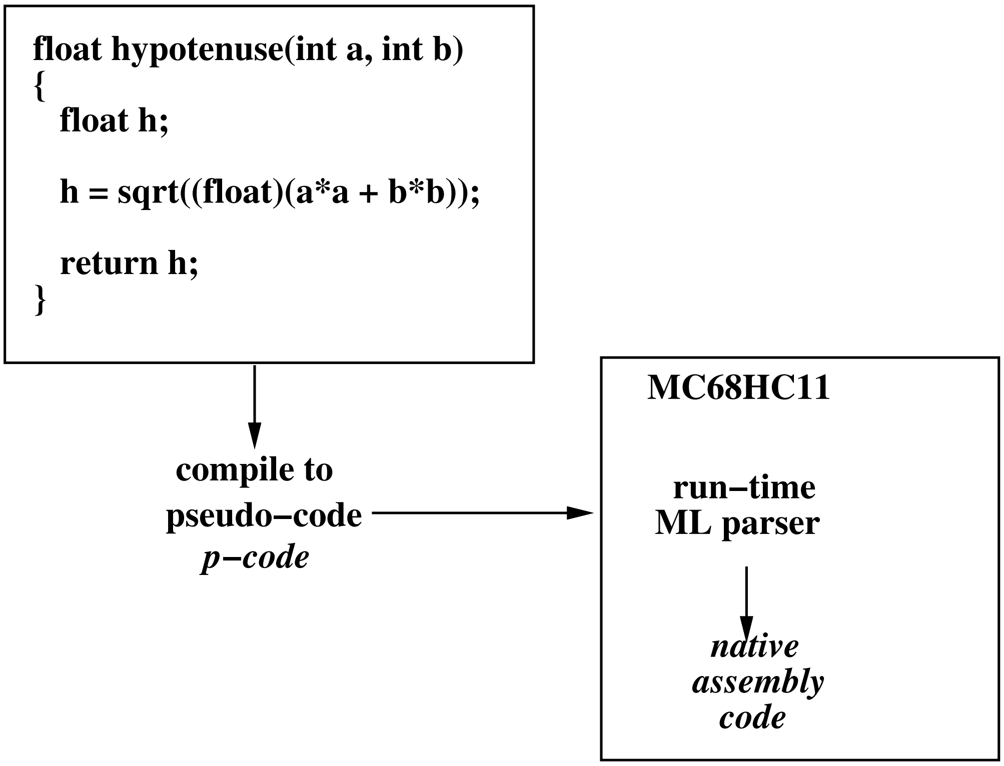
Computational Hardware - Motorola MC68HC11

Specifications - see HandyBoard Technical Reference

- 52-pin, 2 MHz Motorola 6811 microprocessor
- 32K of battery-backed CMOS static RAM
- 4 DC motor drivers (L293D)
- 7 analog and 9 digital inputs



Interactive C



- interpreted pcode - error checking
- portable to other μcontrollers - redo pcode interpreter
- produces small, compact pcode
- compiles to stack machine $\Rightarrow$ supports multi-tasking/scheduling
- at the expense of some speed.

Interactive C Programming - Start Up

- hold STOP at power up - LEDS should light and then go out, this signifies **download mode**
- on host workstation: *dl pcode_hb.s19*
  downloads pcode interpreter
- powercycle the Handy Board and it should be running Interactive C (IC), run *ic* on the host to talk in C to the Handy Board.

C> printf("hello world\n");

$\Downarrow$

produces "hello world" on the LCD display

Interactive C Programming
Function Library

Motor Drivers

- *void fd(int m)* - motor m, "forward" direction (green LED) full power
- *void bk(int m)* - motor m, "backward" direction (red LED) full power
- *void motor(int m, int p)* - pwm controller on motor m, $-100 \leq p \leq 100$ actually selects one of seven discrete duty cycles in the pwm
- *void off(int m)* - motor m off
- *void alloff()* - all motors off
- plus a digital (stepper motor) driver library on PA7

Interactive C Programming
Function Library

Inputs

- *int digital(int p)*: boolean input on sensor port p, sensors are active low so that function *digitial()* returns TRUE (1) for a 0 volt input
- *int analog(int p)*: returns an integer between $0 - 255$ representing the analog input on port p
- *int stop_button()*: return 1 is STOP button is depressed and 0 otherwise.
- *int start_button()*: return 1 is START button is depressed and 0 otherwise.
- *void stop_press()*
- *void start_press()*
- *int knob()*: returns the position of the potentiometer knob between 0 and 255

```
while(!stop_button())
        printf(“%d\n”, knob());
```

Interactive C Programming
Function Library

Infrared, Time, and Tone Functions

Interactive C Programming - IC commands

- *load* $< filename >$ - IC can be used to compile functions defined in $filename$ and download pcode to the Handy Board
  - *load* $< filename.c >$ - compiles and downloads
  - $as11\_ic$ $< filename.asm >$ - creates binary
    load $< filename.icb >$ - downloads to handy board
  - *load* $< listfile.lis >$ - downloads a list of .c and .icb files
- *unload* $< filename >$ - deletes $filename$ functions from the list of active functions, resets processor, and reloads the balance of the list.
- *list functions*
- *list globals*
- *help* and *quit*

Interactive C Programming - Multitasking

process - a function with unique *pid*, local program stack (256 bytes default) for return addresses, local variables, arguments, etc, and processor allocation in milliseconds. Processes communicate through global variables.

process table - a set of processes that are executed in a fixed sequence for a period determined by their processor allocation (default 5 msec)

- $int\ start\_process(function{-}call(...), [TICKS], [STACK{-}SIZE])$ returns the process id
- $int\ kill\_process(int\ pid)$ returns 0 if successfully killed, 1 otherwise
- *kill_all* - stops all running processes
- *ps* - processor status

Interactive C Programming - Multitasking

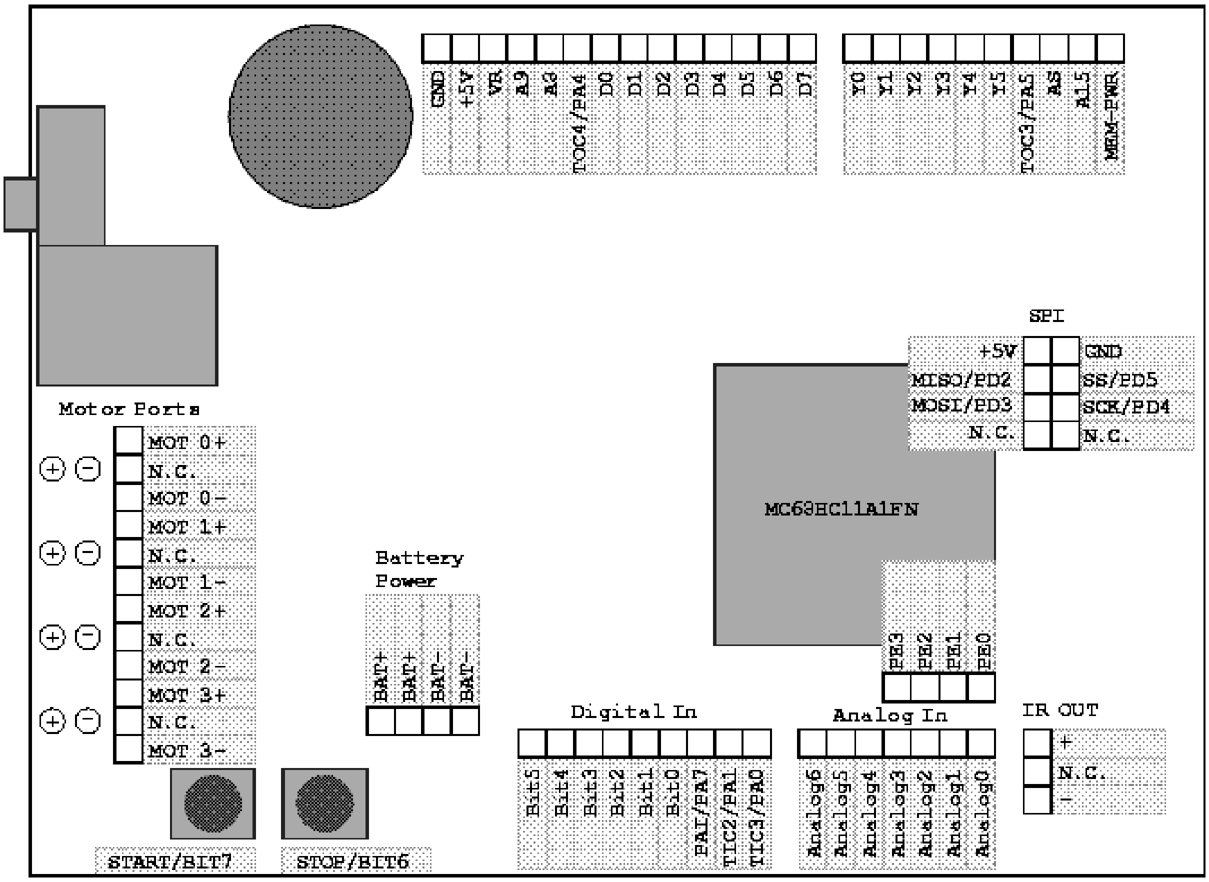
Example

```
void check_sensor(int n) {
   while (1) printf(“sensor %d is %d\n”, n, digital(n));
}
void main()
{  int pid;
   pid = start_process(check_sensor(2));
   sleep(1.0);
   kill_process(pid);
}
```

Q: how often does (*check_sensor*()) actually run?

A: depends what else is in the process table, if nothing else is there, this process runs continuously in 5 *msec* chunks.

Example: IR obstacle Detection



Notes:
The 'Bit X' on the buttons and Digital In are bits of a read to 0x7FFF.

Y0 through Y5 are active lows. This table shows what to do to force them low for a cycle.

Write 0x4000 = Y0
Read 0x4000 = Y1
Write 0x5000 = Y2
Read 0x5000 = Y3
Write 0x6000 = Y4
Read 0x6000 = Y5

Example: IR obstacle Detection

/* file MAIN.C */

```
int run_IR() {
        while(1) sample_IR();                IC can't start ASM processes ???
}

int main() {
        init_IR();
        pid = start_process(run_IR(0));
        return 0;
}
```

/* file IR.ASM */

```
#include "6811regs.asm"

ORG MAIN_START

subroutine_init_IR():
        LDAA #$3C                    /* bits 2, 3, 4, and 5 0011 1100 output */
        STAA DDRD                    /* data direction register for port D */
        CLRA                         /* return 0 */
        RTS


subroutine_sample_IR():
        CLR count
        CLR temp
        LDY #$0A                     /* obstacle hypothesis from 10 samples */
loop:   LDAA #$3C
        STAA PORTD                   /* enable oscillator(s) on PD2, 3, 4, or 5 */
        LDX #200                     /* delay to permit sensor response */


wait_on: DEX
        BNE wait_on

        LDAB $7FFF                   /* read port A digital inputs (bits 5-0) */
        ANDB #$01                    /* assume bit 0 is IR out */
        ADDB count
        STAB temp
```

```
          LDAA #$00
          STAA PORTD           /* disable oscillator(s) on PD2, 3, 4, or 5 */
          LDX #200             /* delay to permit sensor response */

wait_off: DEX
          BNE wait_off

          LDAB $7FFF           /* read port A digital inputs (bits 5-0) */
          ANDB #$01            /* assume bit 0 is IR out */
          EORB #$01
          EORB temp
          ANDB temp
          ADDB count
          STAB count
          DEY
          BNE loop

          LDAA count
          CMPA threshold       /* let's say 6/10 samples =¿ obstacle ? */
          BLT no_obstacle
          LDAA #1
          STAA obstacle?
          BRA fin
no_obstacle:
          CLRA
          STAA obstacle?
          CLRB
fin:      RTS
```

