```
//*********************************************************************
//       Bus Interface module
//*********************************************************************
module bus_iface( addr, as, uds, lds, rw, dtack, w_reg, r_reg );
  input [23:1] addr;
  input as,uds,lds,rw;
  inout dtack;
  output w_reg,r_reg;
  wire ack, gnd, w_reg, r_reg, a,b;

  assign gnd = 0;
  bufif1(dtack, gnd, ack);
  assign ack = w_reg | r_reg;

  assign a = ~(as|uds|lds);
  assign b = addr[23:20] == 'hB;

  assign w_reg = ~rw & a & b;
  assign r_reg = rw & a & b;

endmodule

//*********************************************************************
//       Clock Divider module
//*********************************************************************
module clk_div(pclk, clk);
  input pclk;
  output clk;
  reg [8:0] count;

  always @(posedge pclk)
    count <= count + 1;
  assign clk = count[8];
endmodule

//*********************************************************************
//       Counter module
//*********************************************************************
module cntr(clk,enable,reset,Q);
  input clk,enable,reset;
  output [15:0]Q;
  reg [15:0]Q;

  always @(posedge reset or posedge clk)
  if (reset)
    Q <= 0; //stupid expanding 0
  else
    if (enable)
      Q <= Q + 1;
endmodule

/*Top-Level Wrapper File to implement circuits with the FLEX10K70
on the Ultragizmo board. DO NOT change any of the pin definitions
unless you are completely sure of the consequences.
*/
module wrapper (address, data, codata, srama, sramdl, sramdh, codsync,
                codread, codwrite, sram1en, sram2en, sram1oe, sram2oe,
                sram1we, sram2we, sram1ud, sram2ud, sram1ld, sram2ld,
                br_sfpga, bg_sfpga, bgack_sfpga, clk, fc, berr, as, halt, reset,
                dtack, lds, uds, rw, irqsf, iacksf, hex0, hex1, hex2, hex3,
                sfpga_digital, sfpga_con40, sfpga_logic, sfpga_con60, led, pclk,
                               start, stop, w_reg, r_reg
              );

input [23:1] address;
inout [15:0] data;
input [31:0] codata;
input [17:0] srama;
input [15:0] sramdl;
input [15:0] sramdh;
input codsync, codread, codwrite, sram1en, sram2en, sram1oe, sram2oe;
input sram1we, sram2we, sram1ud, sram2ud, sram1ld, sram2ld;
input br_sfpga, bg_sfpga, bgack_sfpga, clk;
input [2:0] fc;
input berr, as, halt, reset;
inout dtack;
input lds, uds, rw, iacksf;
input [33:0] sfpga_digital;
input [31:0] sfpga_con40;
input [17:0] sfpga_logic;
input [51:0] sfpga_con60;
input pclk;

output [7:0] hex0, hex1, hex2, hex3;
output [15:0] led;
output irqsf;
output start, stop, w_reg, r_reg;

  // Put Code Here
  wire w_reg,r_reg,c_en,d_clk,stp;
  wire [15:0]Q;
  reg start,stop;
  bus_iface bi( address, as, uds, lds, rw, dtack, w_reg, r_reg );
  clk_div   cd( pclk, d_clk );
  cntr      ct( d_clk, c_en, stp, Q );

  assign data = (r_reg) ? Q : 16'bZ;
  assign stp = stop;

  always @(w_reg or sfpga_digital[0])
    if (w_reg)
    begin
      start <= data[0];
      stop  <= data[1];
    end
    else if (sfpga_digital[0])
      stop <= 1;

  assign irqsf = ~stop;
  assign c_en  = ~stop & start;
  assign led[0] = ~stop & start;
  assign led[1] = ~stop & start;
  assign led[2] = ~stop & start;

endmodule

//*********************************************************************
//       Makefile
//*********************************************************************
COMPILE.S=$(CC) $(CFLAGS) $(CPPFLAGS) -c

CFLAGS = -mc68000
LDFLAGS = -lc -lgizmo2 -dc -dp -Ttext 20000 -N
CC = m68k-coff-gcc

AS = m68k-coff-as
AR = m68k-coff-ar
LD = m68k-coff-ld
RANLIB = m68k-coff-ranlib

SREC = /local/bin/m68k-coff-objcopy

CRT0 = crt0.o
```

```
all:    rt_hw.srec

rt_hw.srec:     rt_hw
        ${SREC} -O srec -S  rt_hw  rt_hw.srec

rt_hw: ${CRT0} rt_hw.o cint.o
        ${CC} ${CFLAGS} ${CRT0} rt_hw.o cint.o -o rt_hw ${LDFLAGS}

rt_hw.o: rt_hw.c
        ${CC} ${CFLAGS} rt_hw.c -c

crt0.o: crt0.S
cint.o: cint.S

clean:
        rm -f *.o rt_hw *.srec
```

```
//*********************************************************************
//       poke.h
//*********************************************************************
#define pokeb(a,d) (*(char*)a = (char)d)
#define pokew(a,d) (*(short*)a = (short)d)
#define peekw(a)   (*(short *)a)
#define pokel(a,d) (*(int*)a = (int)d)
```

```
//*********************************************************************
//       rt_hw.c
//*********************************************************************
#include <stdio.h>
#include "poke.h"

/*
 CFG_PORT:  controls the START and STOP flip-flops
 TIME_PORT: read the reaction time from this port
 */
#define CFG_PORT   0xB00000
#define TIME_PORT  0xB00000

#define ICR        0xFFFFFFFA
#define CFPGA_ICR  0x00C20000
#define PDATA      0xFFFFFFF0
#define AUTO6      0x00000078

#define MODE 'a'

extern int intrstub( void );
extern int spl0( void ), spl7( void );

int main( )
{  /*
     Setup the interrupts on the Ultragizmo board
    */
   spl7( );
   pokeb( CFPGA_ICR, 0x06 );
   pokel( AUTO6    , intrstub );
   pokew( ICR      , 0x60ff );
   pokew( PDATA    , 0xfff7 );
   spl0( );
   /*
     Add: Code to initiate test.
    */
   pokew( CFG_PORT, 0x0001 ); //start = 1, stop = 0
   for(;;); /* Wait forever */
}

int interrupt( )
{
   unsigned short time;
   time = peekw( TIME_PORT );

   pokew( CFG_PORT, 0x0000 ); //start=0, stop=0
   if (MODE == 'a')
     printf("Time: %d\n",time);
   else
   {
     char result[6] = "00000";
     char i;
     //hardcoded five digit number conversion
     for (i=5; i>=0; i--)
     {
       result[i] += time % 10;
       time /= 10;
     }
     puts("Time: ");puts(result);puts("\n");
   }
   /*
     Add: Code to
       1. clear interrupt,
       2. print out time
          a. a version that uses printf
          b. a version that does not use the printf family of library functi
             you may use the puts function to output a string
    */
}
```