```
/* --- MODULE.C       Version 1.0
   --- This is E-PBO example module code for Virtex-4/5 FPGA & AVR.
   --- It can be used as a template for creating your own modules.
   --- Search on "module" and replace with your module name.
   --- */

#include <stdio.h>
#include <stdlib.h>      /* required for malloc() */
#include "pbort.h"
#include "module.h"

/* These user options for the sbsSet(), sbsGet(), sbsEvaluator() and
sbsEstimator() routines must appear in the module.h header file*/
#define  MODULE_USER1         101
#define  MODULE_USER1_PTR 102
#define  POWER                201

/* module Local Structure, appear in module.h  */
typedef struct{
  /*put any local variables your module needs to operate here*/
      uint8_t              *inPtr;
      uint8_t              *outPtr;
      uint8_t              nodeNum;
      uint8_t              funcPerfValue;
      int                  power;
      int                  commuRSSI;
      uint8_t              *estInPtr;
      uint8_t              *estOutPtr;
      char                 user1;
} module_localT;

/*module_on             Start up the module.*/
char module_on(processT *p_ptr)
{
      module_localT  *local = (module_localT *)p_ptr->local;

  /*put any code here your module needs when it first turns on*/

      return I_OK;
}

/*module_set            sbsSet() user-defined functions.*/
char module_set(processT *p_ptr, int16_t type, int16_t arg, void *vptr)
{
      module_localT  *local = (module_localT *)p_ptr->local;

      switch(type){
            case MODULE_USER1:
            /* set the value of user1 in the local structure */
                  /* user-defined code */
                  local->user1 = (uint8_t *)vptr;
                  return I_OK;
                  break;
             /*put any code here your module needs to be set*/
            default:
                  return I_ERROR;
      }

      return I_OK;
}

/*module_get            sbsGet() user-defined functions.*/
char module_get(processT *p_ptr, int16_t type, int16_t arg, void *vptr)
{
      module_localT  *local = (module_localT *)p_ptr->local;

      switch(type){
            case MODULE_USER1:
            /* set the value of user1 in the local structure */
                  /* user-defined code */
                  local->user1 = (uint8_t *)vptr;
                  return I_OK;
                  break;
             /*put any code here your module needs to be get*/
            default:
                  return I_ERROR;
      }

      return I_OK;
}

/*module_get            sbsGet() user-defined functions.*/
char module_get(processT *p_ptr, int16_t type, int16_t arg, void *vptr)
{
      module_localT  *local = (module_localT *)p_ptr->local;

      switch(type){
            case MODULE_USER1:
            /* set the value of user1 in the local structure */
                  /* user-defined code */
                  local->user1 = (uint8_t *)vptr;
                  return I_OK;
                  break;
             /*put any code here your module needs to be get*/
            default:
                  return I_ERROR;
      }

      return I_OK;
}

/*module_func           module functionality*/
void module_func(module_localT *ptr, void *inPort, void *outPort)
{
      /* This is the main function. Put the code you want to execute
      periodically in here. */
}

/*module_cycle          Process module information.*/
char module_cycle(processT *p_ptr)
{
      module_localT  *local = (module_localT *)p_ptr->local;

       /*call module_func here using PBO in/out ports to bind*/
      module_func(local, (void *)local->inPtr, (void *)local>outPtr);

       return I_OK; /*can return SBS_OFF if the module shuts itself off */

}

/*module_eval         Evaluate module performance (func & non-func).*/
char module_eval(processT *p_ptr, int type, int arg, void *vptr)
{
      module_localT  *local = (module_localT *)p_ptr->local;

      switch(type){
            case POWER:
            /* evaluate power consumption */
                  /* user-defined code */
                  *((uint8_t *)vptr) = local->power;
                  return I_OK;
                  break;
             /*put any code here user wants to evaluate*/
            default:
                  return I_ERROR;
      }

      return I_OK;
}

/*module_est          Estimate module performance (Only func).*/
char module_est(processT *p_ptr)
{
      module_localT  *local = (module_localT *)p_ptr->local;

      /*call module_func here using PBO estIn/estOut ports to bind*/
      module_func(local, (void *)local->estInPtr, (void *)local>estOutPtr);

      return I_OK;
}

/*module_init           Initialize the module*/
char  module_init(processT *p_ptr, void*vptr)
{
      /* intialize the function pointers*/
      p_ptr->on_fptr = module_on;
      p_ptr->set_fptr = module_set;
      p_ptr->get_fptr = module_get;
      p_ptr->cycle_fptr = module_cycle;
      p_ptr->off_fptr = NULL; /*If user defines modle_off, initialize it*/
      p_ptr->eval_fptr = module_eval;
      p_ptr->est_fptr = module_est;

      /*Allocate the local structure for the module*/
      if((p_ptr->local = (pointer)malloc(sizeof(ssd_localT))) == NULL){
                  return I_ERROR;
      }

      /* define a pointer points to local structure */
      module_localT  *local = (module_localT *)p_ptr->local;

      /* Initialize the local structure (module dependent) */
      local->user1 = 0;
      local->power = 20;

      /* put any initialization here */

      return I_OK;
}
```