Software en Tiempo Real

Msc. Ing. Carlos Centeno
Ingeniería Electrónica
UTN FRC

Año 2022

Modulo de I/O

- Sistema de Monitoreo y Control
  - Entradas Discretas
  - Salidas Discretas
  - Modulo de E/S

Entradas Digitales

- En un sistema embebido disponemos en general de entradas digitales para monitorear:

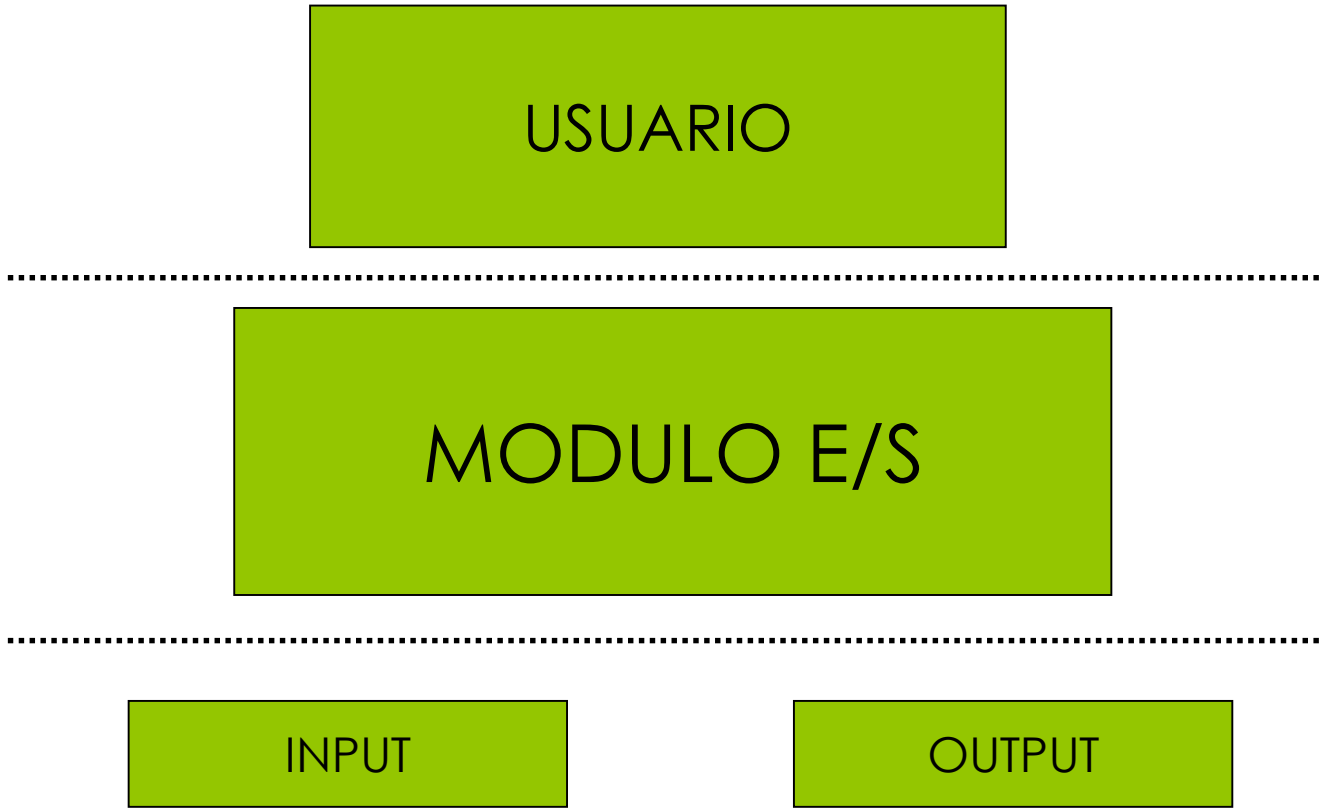
FINALES de carrera
SWITCHS de presión
CONTACTOS varios
DETECTORES de proximidad

Salidas Digitales

- En un sistema embebido disponemos en general de salidas digitales para controlar

CONTACTORES varios
ALARMAS sonoras
ALARMAS visuales
MOTORES
VENTILADORES

Modulo I/O

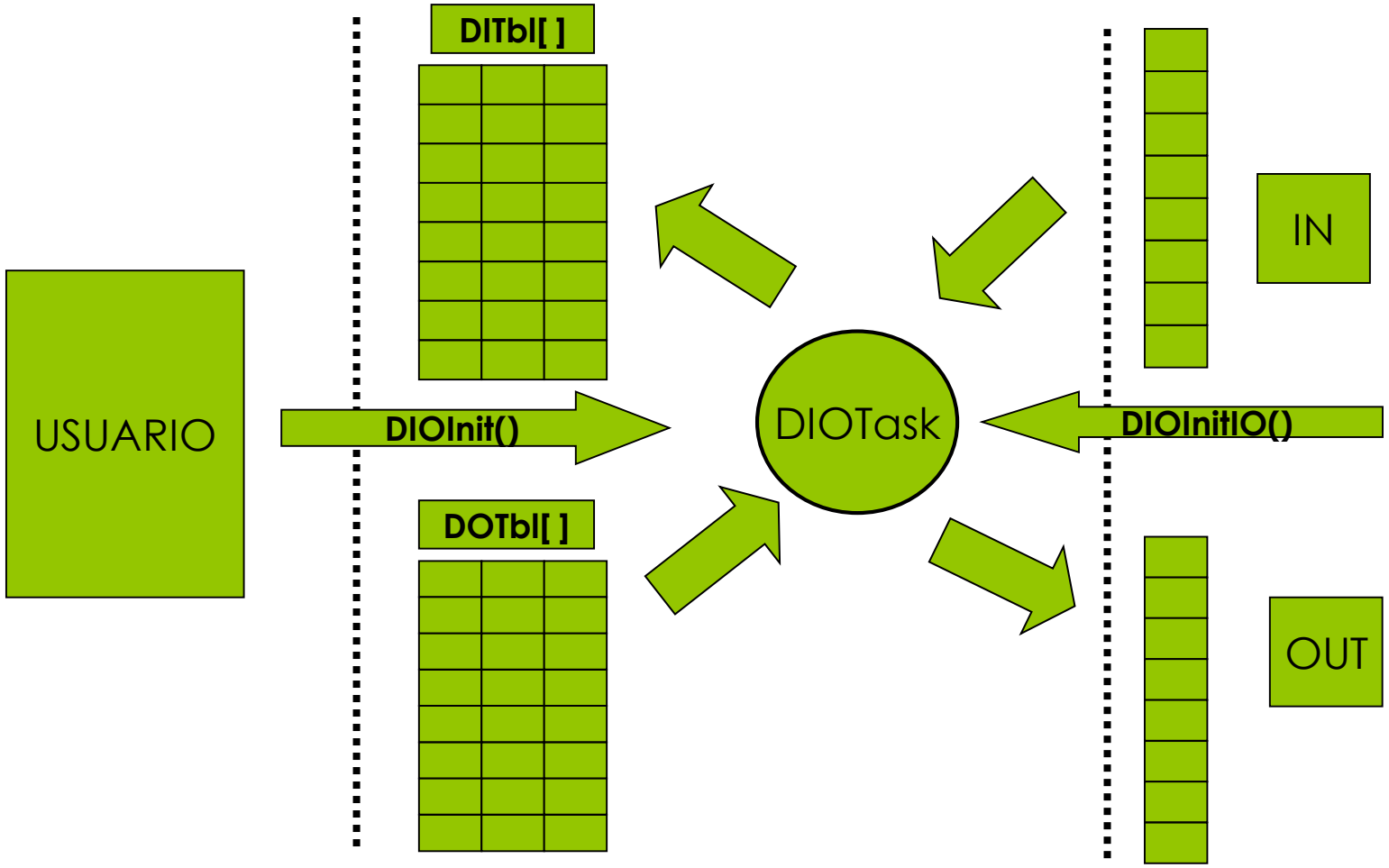


Modulo I/O

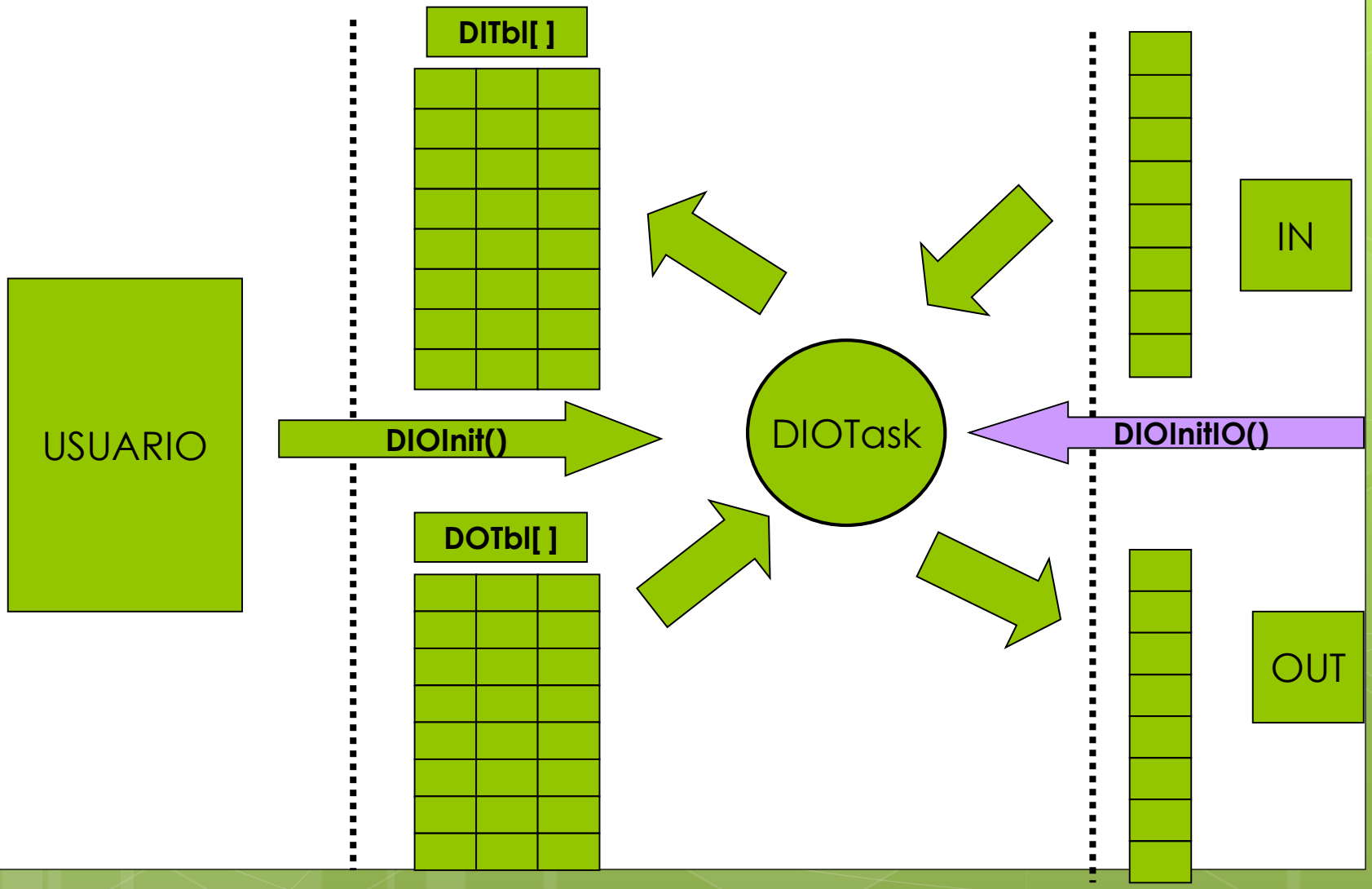
- Ventajas:
- Independencia del HARDWARE de salida
  - PIC :
    - PORTB = 234
  - PC:
    - Outport (0x300, 234)

- Si cambiamos de plataforma, debemos modificar el código con el cual se accede a los periféricos.
- En el caso de emplear un modulo de E/S, solo es necesario actualizar este DRIVER al cambiar de plataforma.

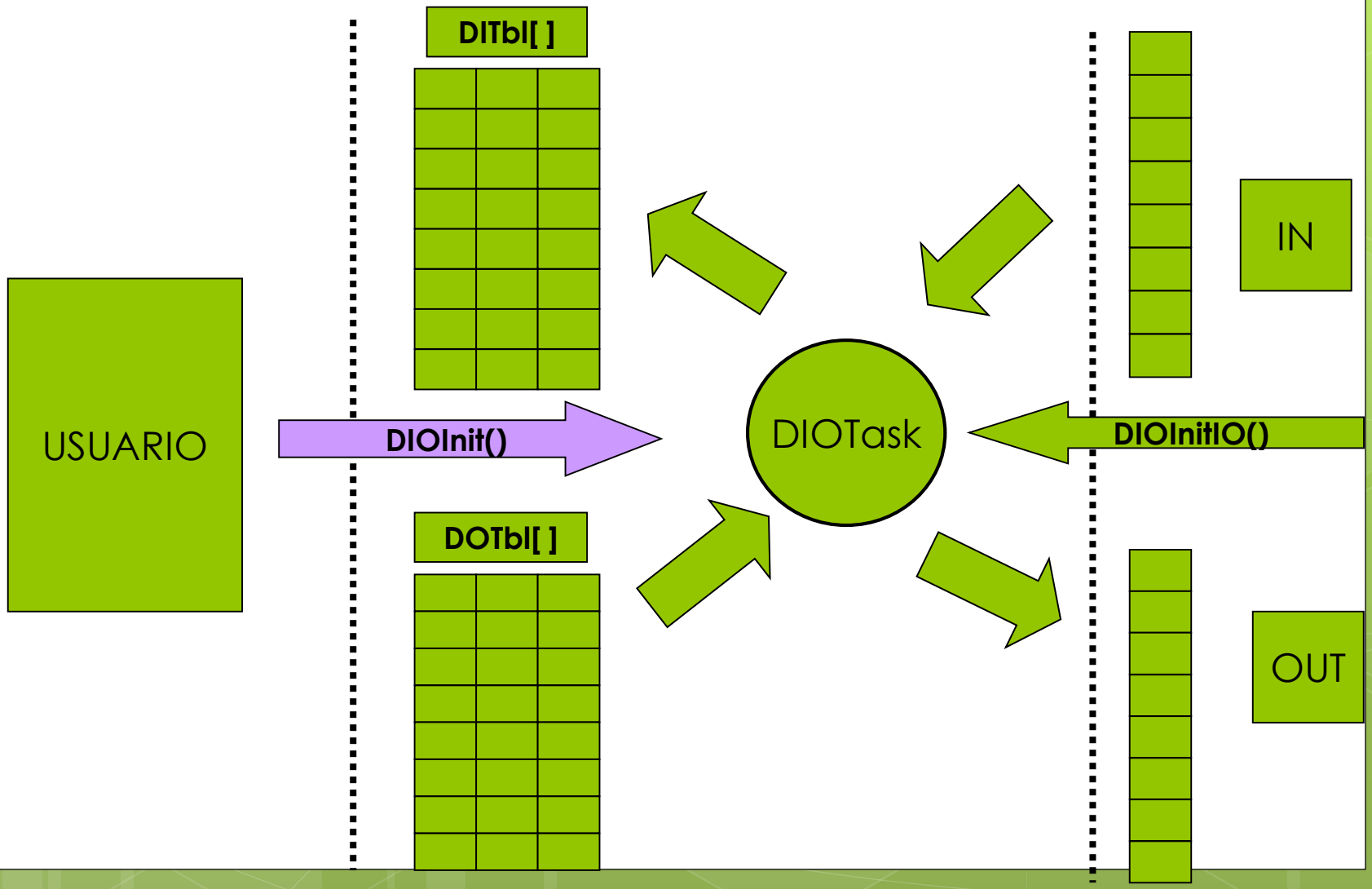
Modulo I/O



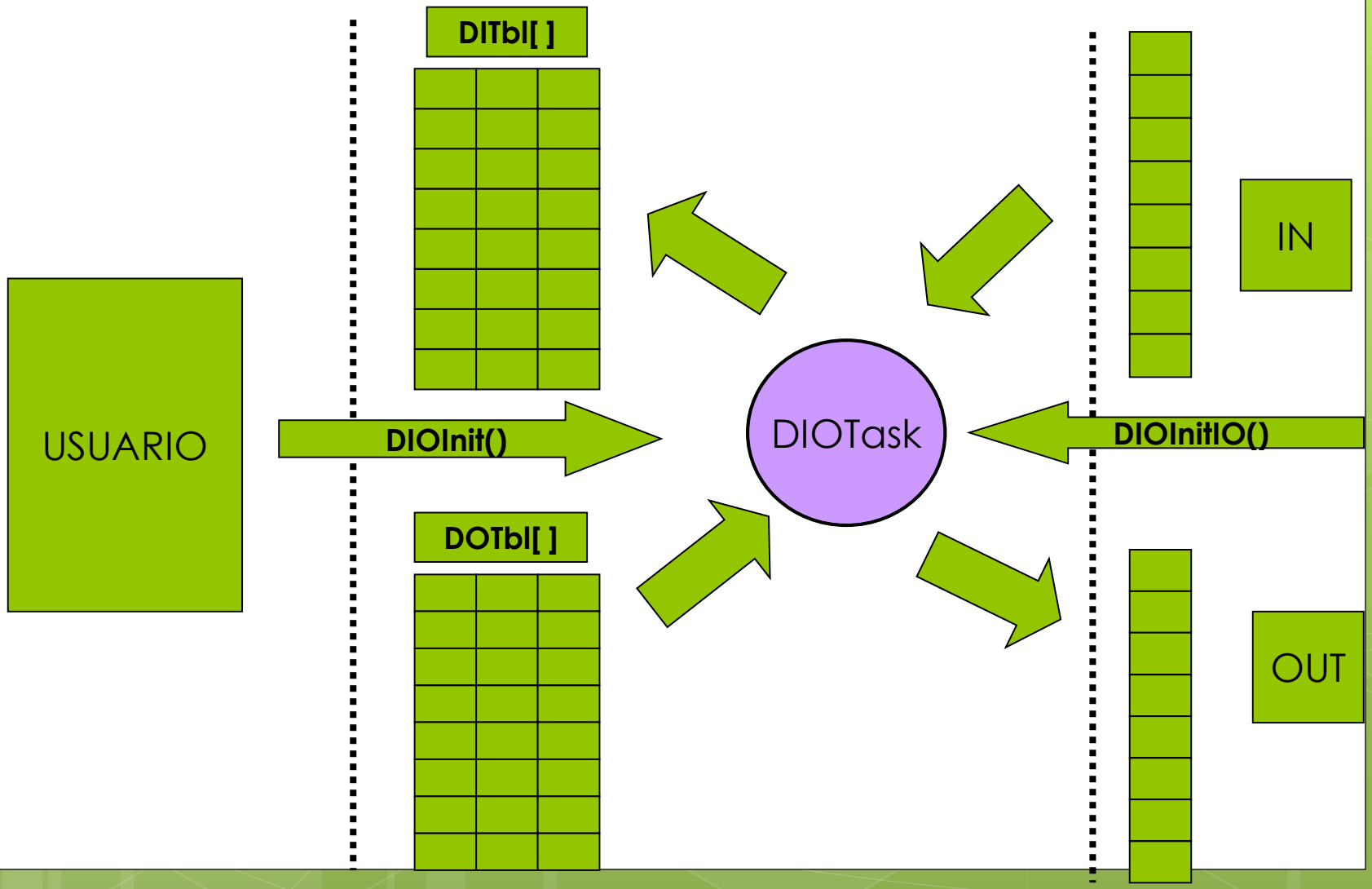
Modulo I/O



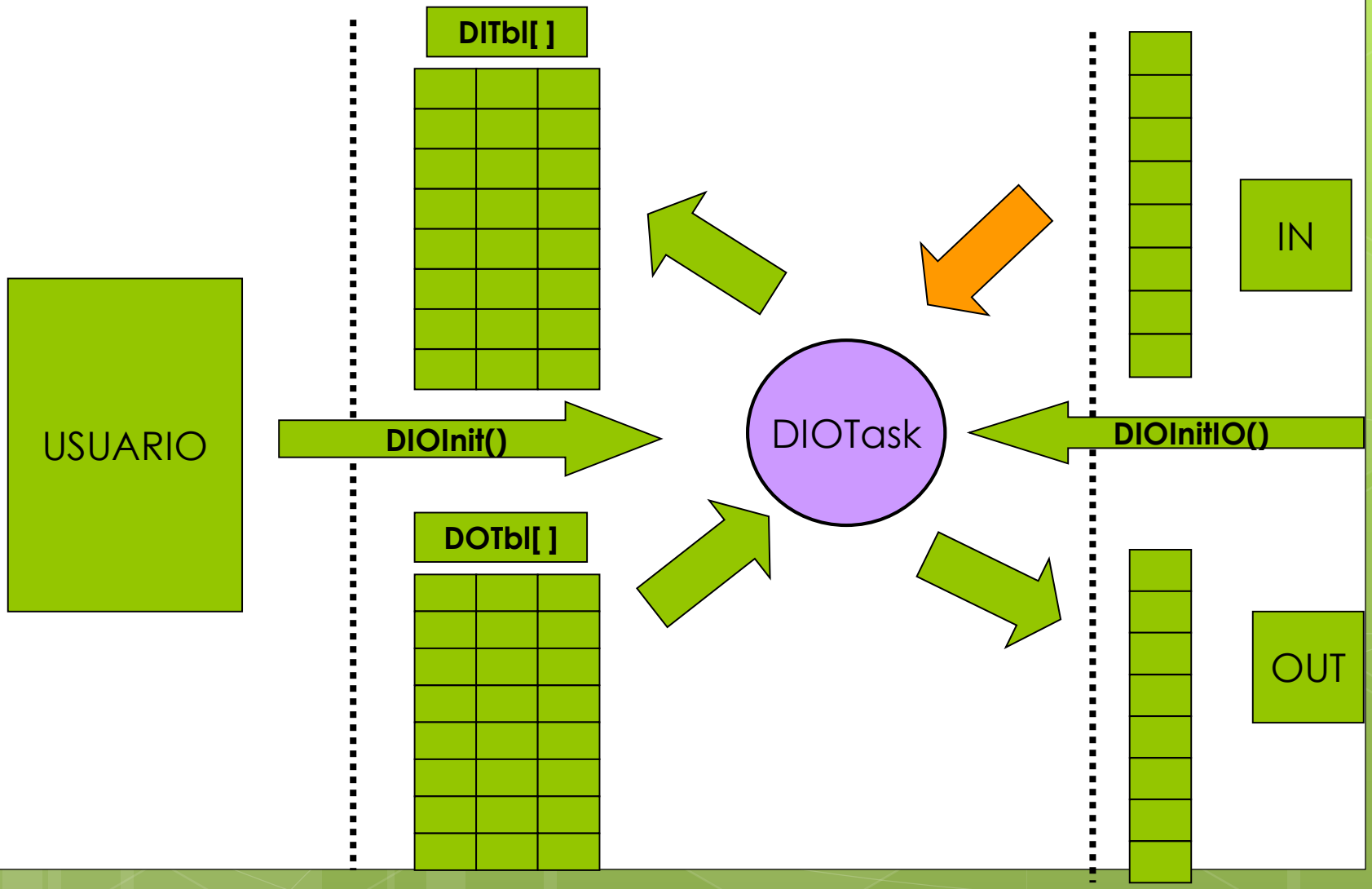
Modulo I/O



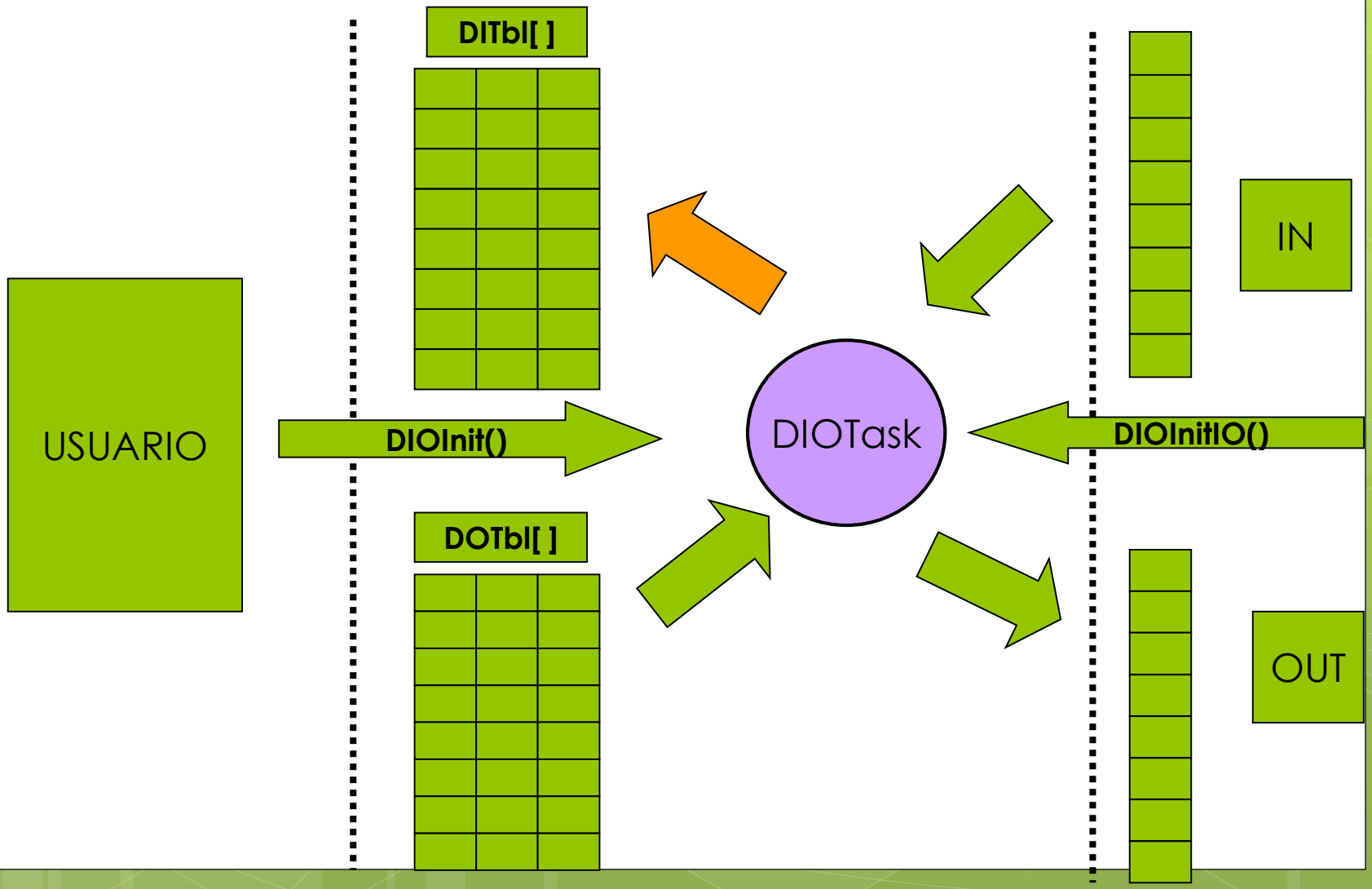
Modulo I/O



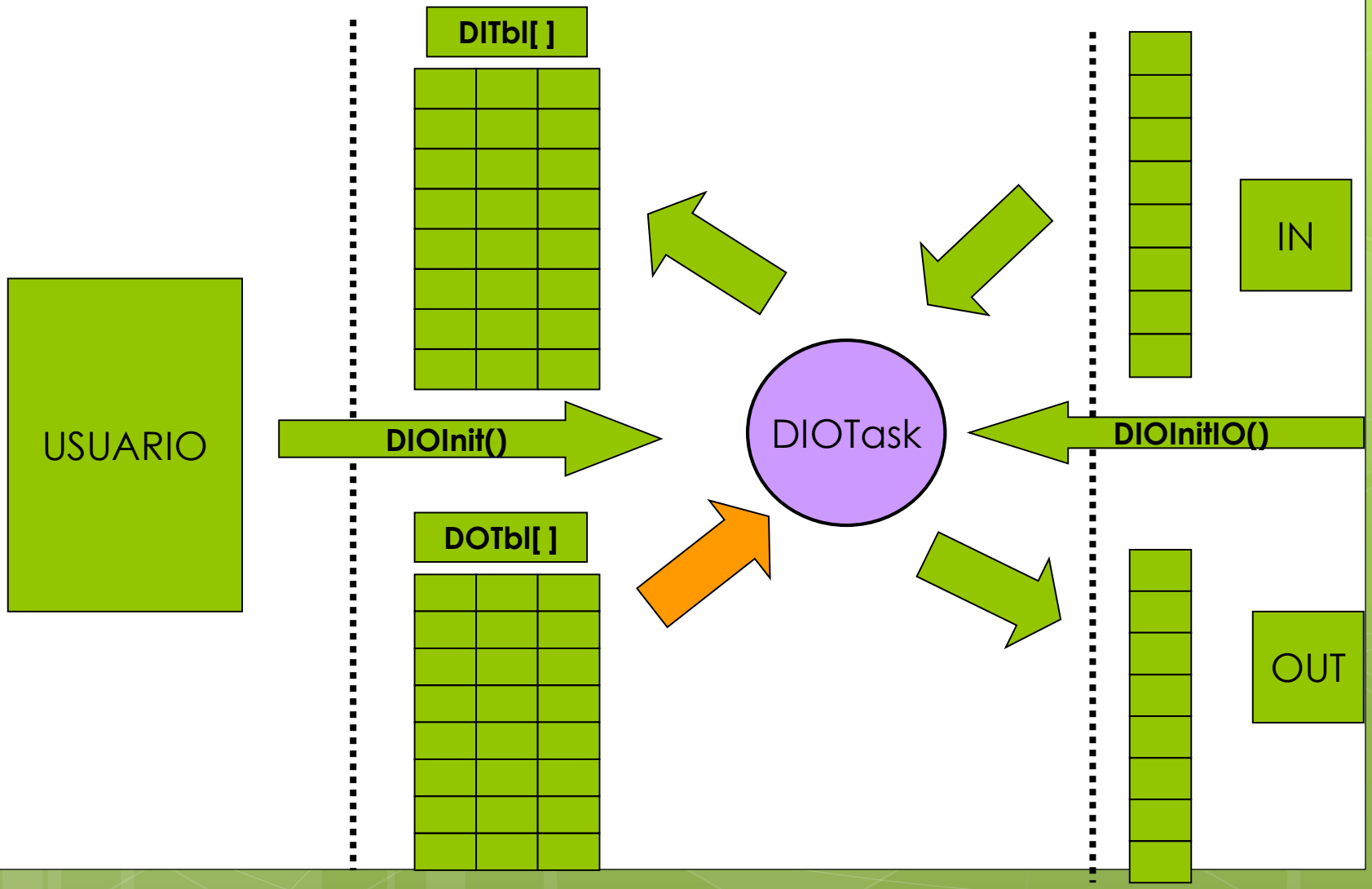
Modulo I/O



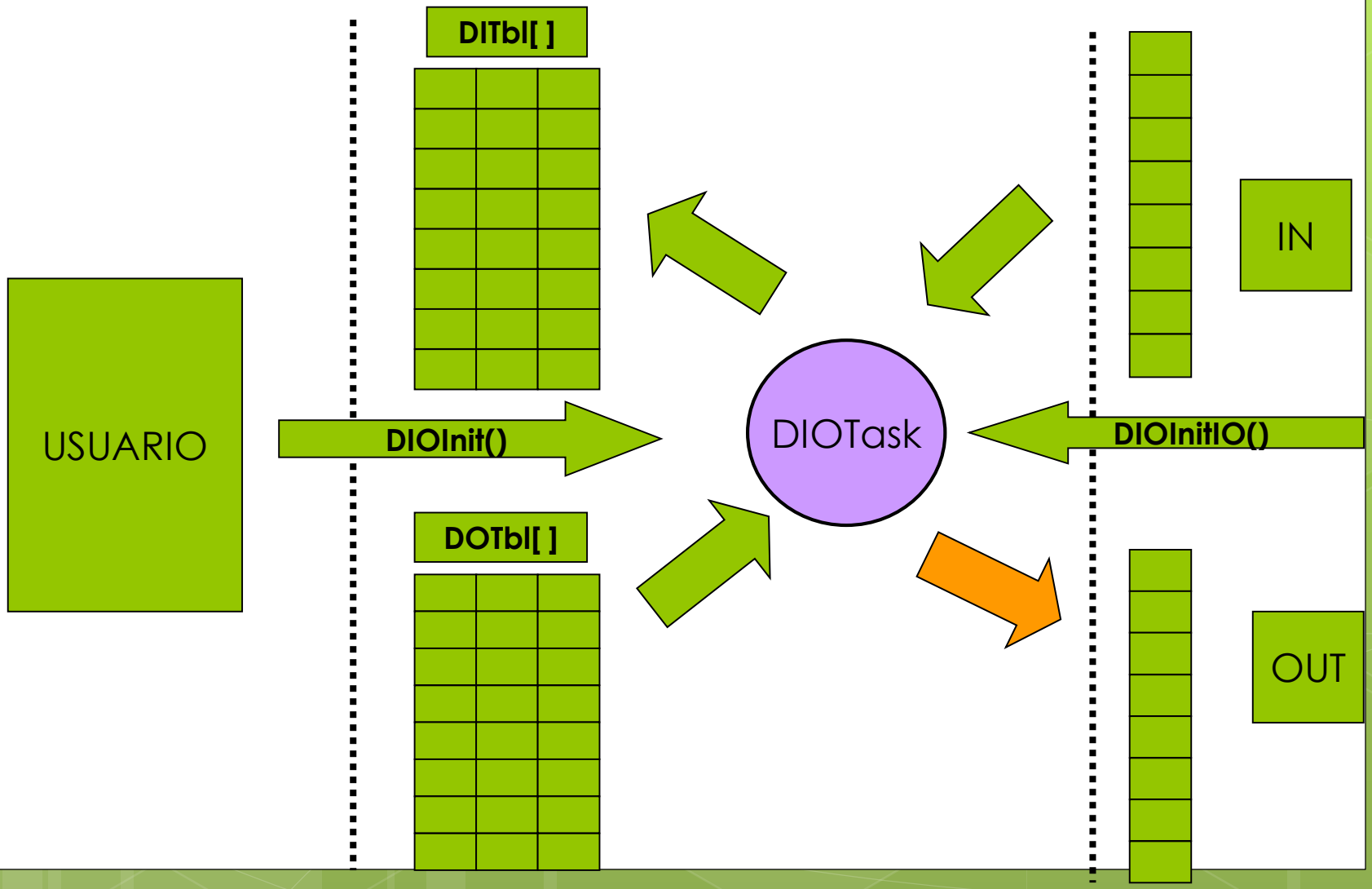
Modulo I/O



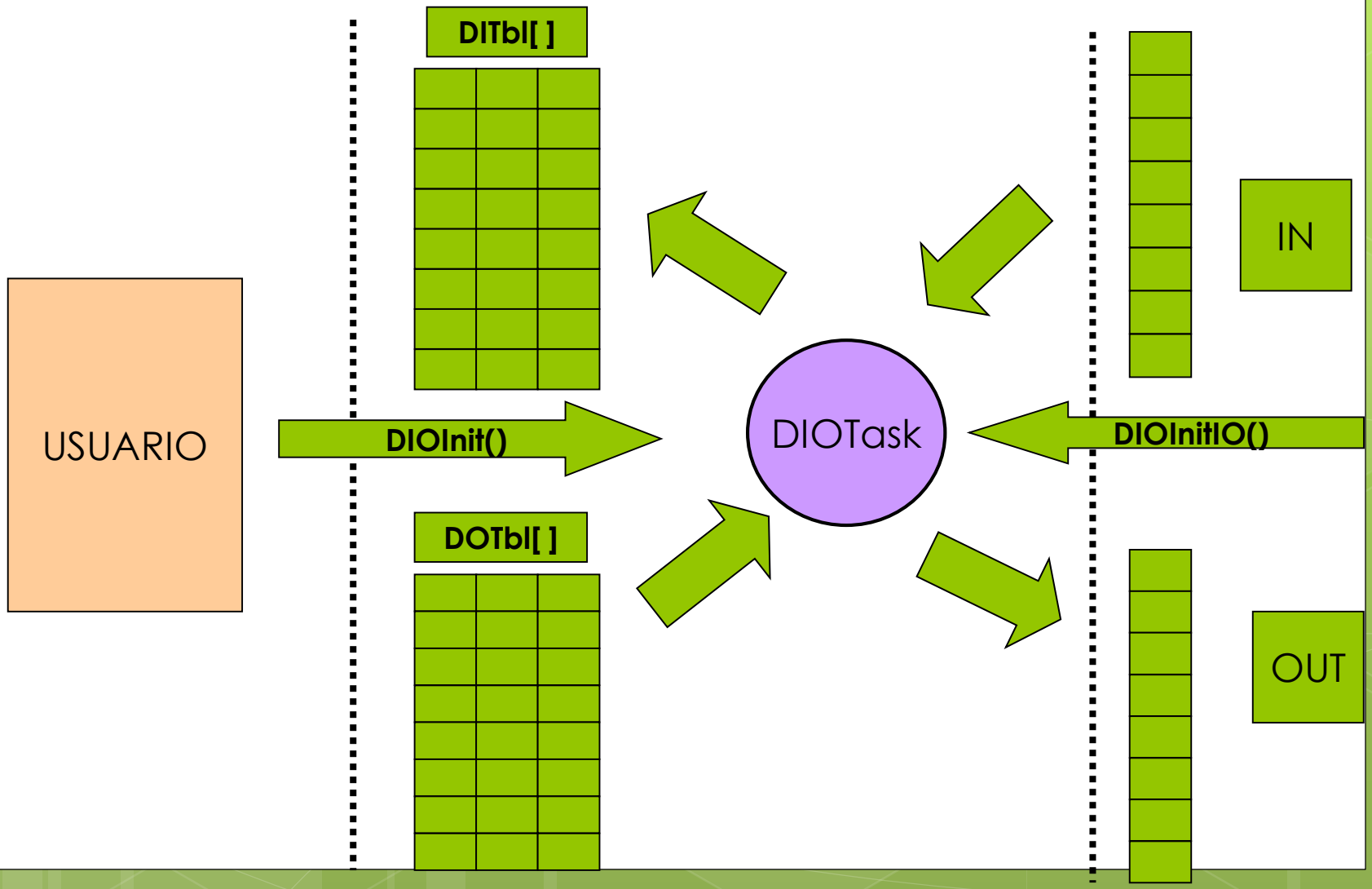
Modulo I/O



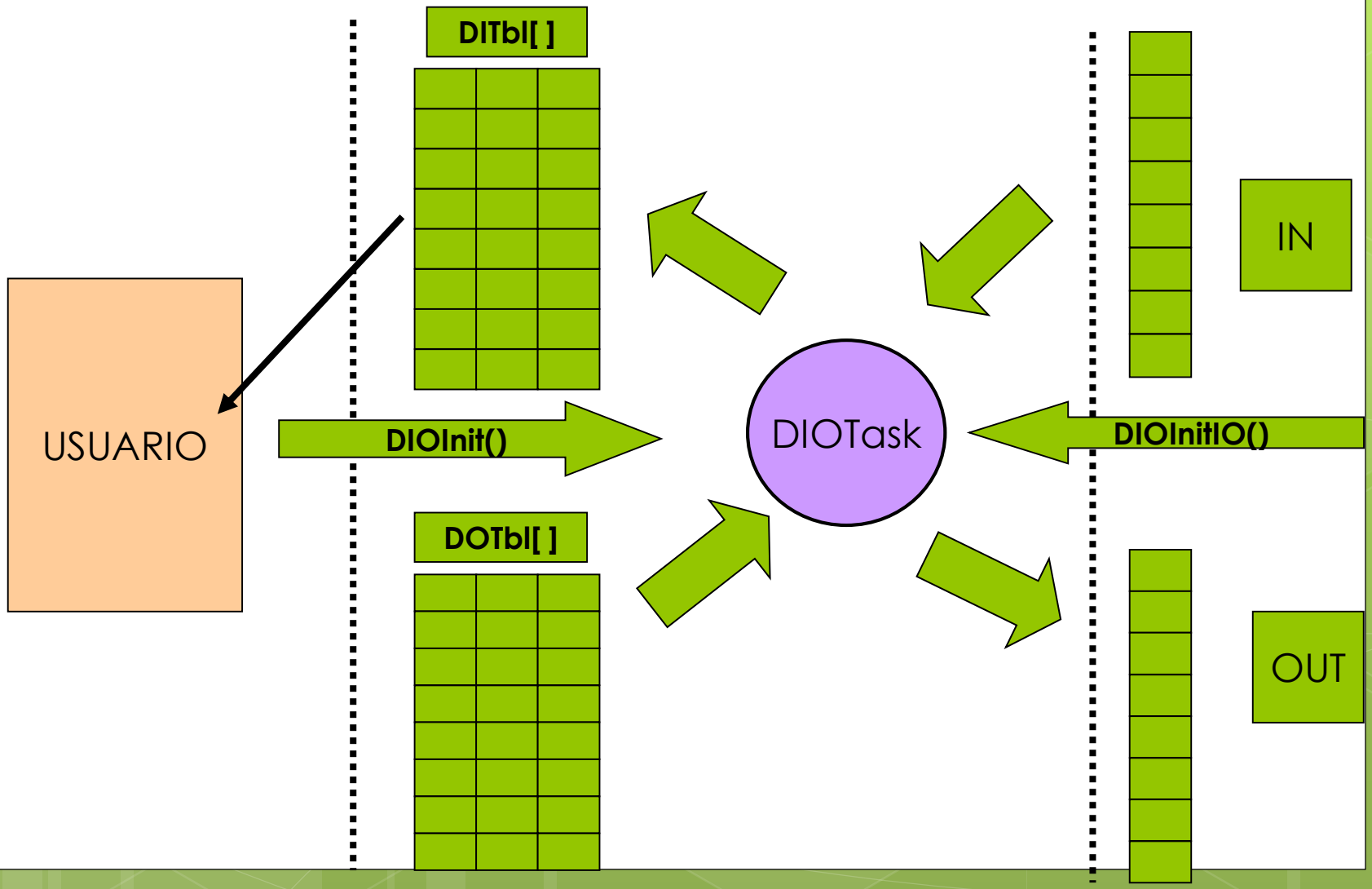
Modulo I/O



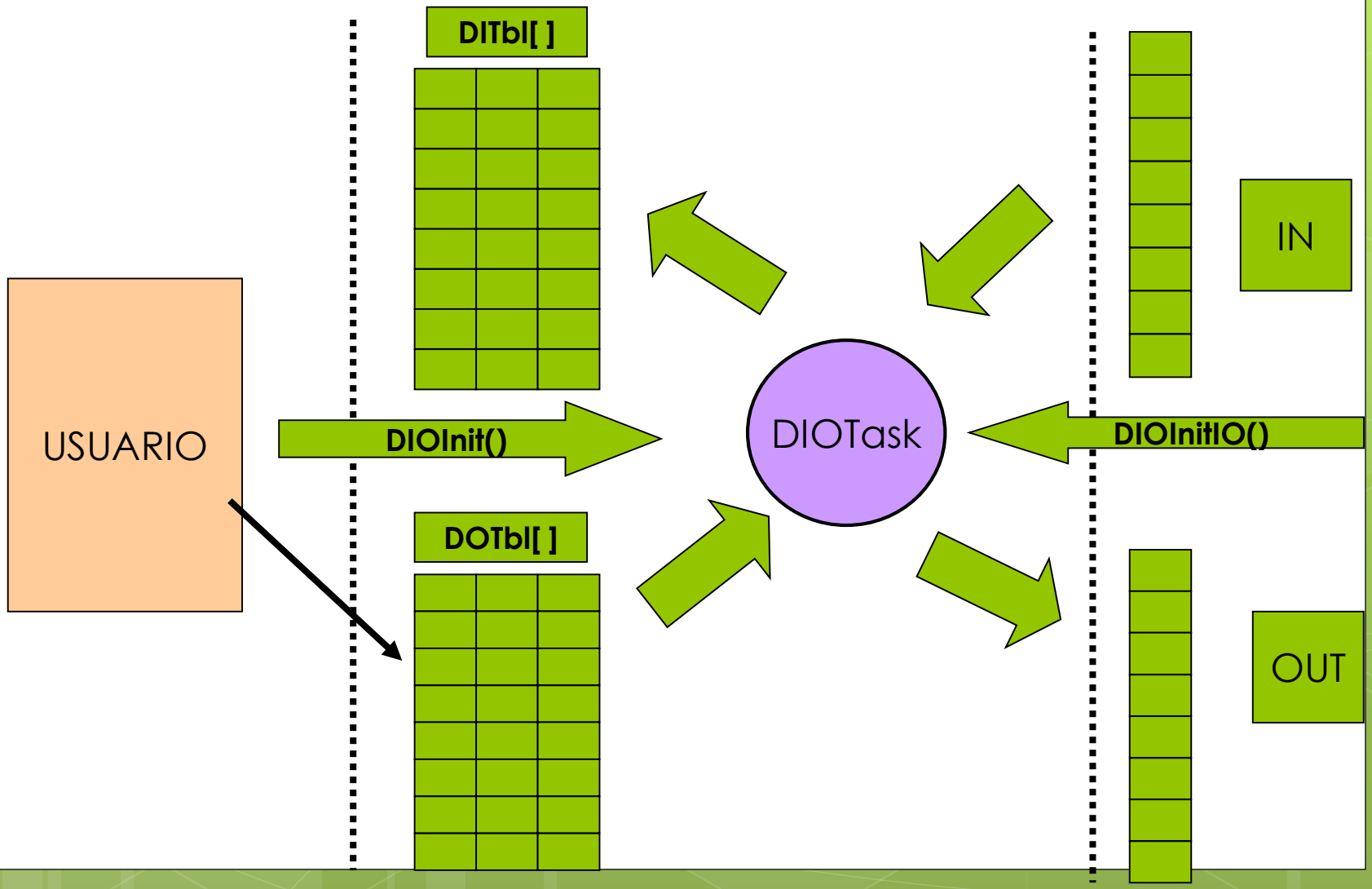
Modulo I/O



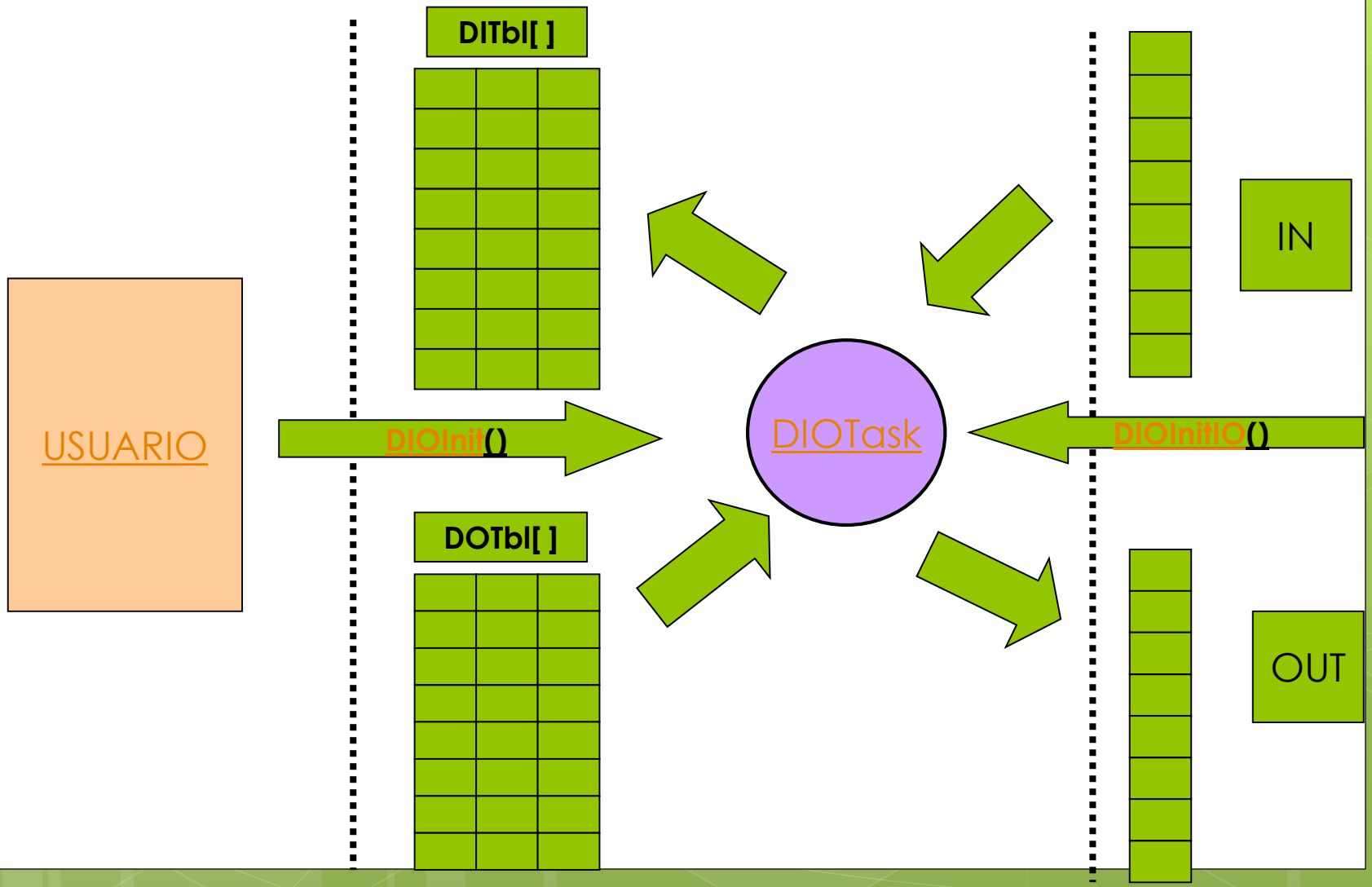
Modulo I/O



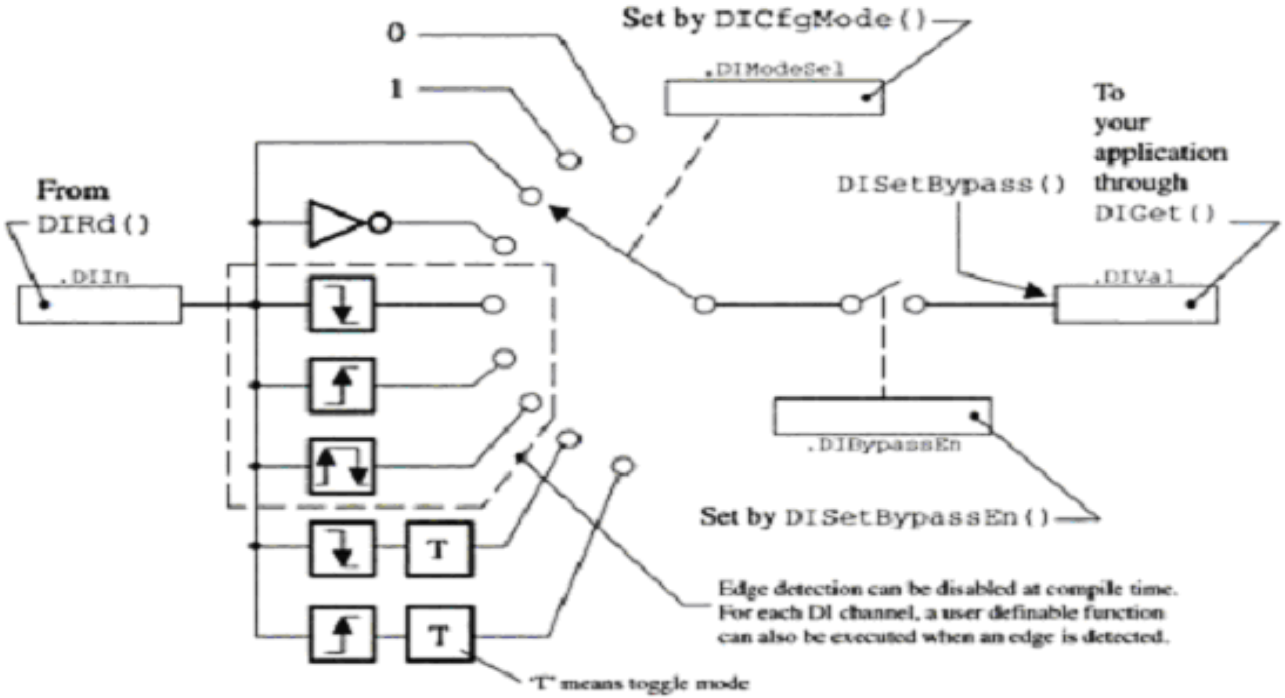
Modulo I/O



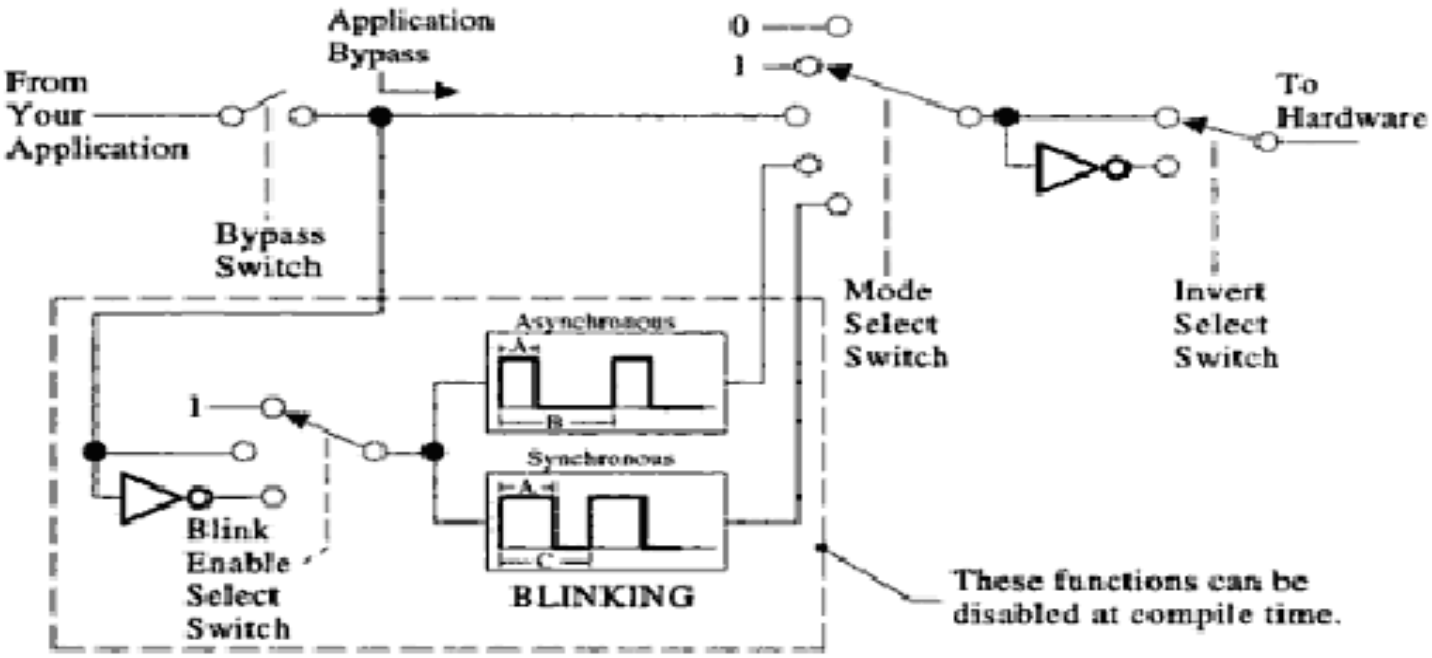
Análisis de Cada Parte



Canal de Entrada DISCRETO



Canal de Salida DISCRETO



DIRd()

- Lee el PUERTO FISICO:

```
void  DIRd (void)
{
    DIO_DI  *pdi;
    INT8U    i;
    INT8U    in;
    INT8U    msk;

    pdi = &DITbl[0];                          /* Point at beginning of discrete inputs
     */
    msk = 0x01;                              /* Set mask to extract bit 0              */
    in  = inp(0x0301);                        /* Read the physical port (8 bits)
     */
    for (i = 0; i < 8; i++) {                /* Map all 8 bits to first 8 DI channels
     */
        pdi->DIIn   = (BOOLEAN)(in & msk) ? 1 : 0;
        msk       <<= 1;
        pdi++;
    }
}
```

VOLVER

DOWr()

- Escribe en el Puerto Físico

```
void  DOWr (void)
{
    DIO_DO  *pdo;
    INT8U    i;
    INT8U    out;
    INT8U    msk;

    pdo = &DOTbl[0];                    /* Point at first discrete output channel          */
    msk = 0x01;                         /* First DO will be mapped to bit 0                */
    out = 0x00;                         /* Local 8 bit port image                          */
    for (i = 0; i < 8; i++) {           /* Map first 8 DO to 8 bit port image              */
        if (pdo->DOOut == TRUE) {
            out |= msk;
        }
        msk <<= 1;
        pdo++;
    }
    outp(0x0300, out);                  /* Output port image to physical port              */
}
```

VOLVER

DIUpdate()

```
static  void  DIUpdate (void)
{
    INT8U    i;
    DIO_DI  *pdi;

    pdi = &DITbl[0];
    for (i = 0; i < DIO_MAX_DI; i++) {
        if (pdi->DIBypassEn == FALSE) {          /* See if discrete input channel is bypassed          */
            switch (pdi->DIModeSel) {            /* No, process channel                                */
                case DI_MODE_LOW:                /* Input is forced low                                */
                    pdi->DIVal = 0;
                    break;
                case DI_MODE_HIGH:               /* Input is forced high                               */
                    pdi->DIVal = 1;
                    break;
                case DI_MODE_DIRECT:                 /* Input is based on state of physical input      */
                    pdi->DIVal = (INT8U)pdi->DIIn;   /* Obtain the state of the sensor                */
                    break;
                case DI_MODE_INV:                /* Input is based on the complement state of input    */
                    pdi->DIVal = (INT8U)(pdi->DIIn ? 0 : 1);
                    break;
```

VOLVER

DOUpdate()

```
static  void  DOUpdate (void)
{
    INT8U     i;
    BOOLEAN   out;
    DIO_DO   *pdo;

    pdo = &DOTbl[0];
    for (i = 0; i < DIO_MAX_DO; i++) {          /* Process all discrete output channels              */
        if (pdo->DOBypassEn == FALSE) {         /* See if DO channel is enabled                      */
            pdo->DOBypass = pdo->DOCtrl;        /* Obtain control state from application             */
        }
        out = FALSE;                            /* Assume that the output will be low unless changed */
        switch (pdo->DOModeSel) {
            case DO_MODE_LOW:                   /* Output will in fact be low                        */
                break;
            case DO_MODE_HIGH:                  /* Output will be high                               */
                out = TRUE;
                break;
            case DO_MODE_DIRECT:                /* Output is based on state of user supplied state   */
                out = pdo->DOBypass;
                break;
```

VOLVER

DIOInit()

```
void  DIOInit (void)
{
    INT8U    err;
    INT8U    i;
    DIO_DI  *pdi;
    DIO_DO  *pdo;


    pdi = &DITbl[0];
    for (i = 0; i < DIO_MAX_DI; i++) {
        pdi->DIVal         = 0;
        pdi->DIBypassEn    = FALSE;
        pdi->DIModeSel     = DI_MODE_DIRECT;     /* Set the default mode to direct input
     */
#if DI_EDGE_EN
        pdi->DITrigFnct    = (void *)0;          /* No function to execute when transition
     detected    */
        pdi->DITrigFnctArg = (void *)0;
#endif
        pdi++;
    }
```

DIOInit()

```
 pdo = &DOTbl[0];
   for (i = 0; i < DIO_MAX_DO; i++) {
       pdo->DOOut        = 0;
       pdo->DOBypassEn   = FALSE;
       pdo->DOModeSel    = DO_MODE_DIRECT;      /* Set the default mode to direct output      */
       pdo->DOInv        = FALSE;
#if DO_BLINK_MODE_EN
       pdo->DOBlinkEnSel = DO_BLINK_EN_NORMAL;  /* Blinking is enabled by direct user         */
       pdo->DOA          = 1;
       pdo->DOB          = 2;
       pdo->DOBCtr       = 2;
#endif
       pdo++;
   }
#if DO_BLINK_MODE_EN
   DOSyncCtrMax = 100;
#endif
   DIOInitIO();
   OSTaskCreate(DIOTask, (void *)0, &DIOTaskStk[DIO_TASK_STK_SIZE], DIO_TASK_PRIO);
}
```

VOLVER

DIOInitIO()

```
void  DIOInitIO (void)
{
   outp(0x0303, 0x82);          /* Port A = OUT, Port B = IN, Port C = OUT          */
// ACA configuro el HARDWARE correspondiente
}
```

VOLVER

DIOTask()

```
static  void  DIOTask (void *data)
{
    data = data;
    for (;;) {
        OSTimeDly(DIO_TASK_DLY_TICKS);

 /* Delay between execution of DIO manager        */


        DIRd();                          /* Read physical inputs and map to DI
     channels   */
        DIUpdate();                       /* Update all DI channels                    */
        DOUpdate();                       /* Update all DO channels                    */
        DOWr();                          /* Map DO channels to physical outputs
     */
    }
}
```

VOLVER

Funciones Usuario

- **DICfgMode()**
- **DOCfgMode()**
- **DOSet()**
- **DOGet()**
- **DIGet()**

VOLVER