Chapter 4

SOES EtherCAT Slave Stack

1 Overview

The SOES is a library that provides user applications with means to access the EtherCAT fieldbus communication environment:

- EtherCAT State Machine
- Mailbox Interfaces
- Protocols
  - CoE
  - FoE

Support for mailbox and protocols are typical examples when you need a slave stack to control the Application Layer of EtherCAT. The PDI used for such applications are either SPI (or similar) or CPU Interface

2 Get started

Goto `SOES on GitHub`, there you can find releases or current version.

Clone or download to the destination folder. SOES is split in 3 parts.

- soes - the generic part configured from the application
- application - the application API that configure the stack and control the execution
- soes HAL - hardware abstraction layer for the slave stack, not generic and complements the application configuration of the stack by providing necessary hardware configuration and functions

For a project, use generic soes and pick one application and soes/hal as template that fits hardware and software platform. If the EtherCAT Slave Editor is used it will provide parts of the application generated, otherwise they have to be written.

Project structure

- soes archive

  - soes
  - soes/include/sys/gcc/cc.h
- application binary
  - application
  - soes/hal

Under application a set of sample applications can be found as reference designs on different platforms.

3 Initialization the EtherCAT slave stack

The EtherCAT Slave stack is setup with an ESC (EtherCAT Slave Controller) generic initialization function ecat_slv_init. The esc_cfg_t argument passed to the stack is copied, except the user_arg, to stack internal variables and the esc_cfg_t arg can go out-of-scope.

There are 3 modes of operations for the stack

- Polling (Free-Run)
  - PDI interrupt is not enabled
  - SyncX interrupt are not enabled
  - AL Event are polled for events every ESC Read/Write to be handled by the stack
  - ecat_slv() get called regular to handle stack operations
- Mixed Polling/Interrupt (SM or DC Synchronous)
  - PDI interrupt is enabled
  - Only SM2 should be masked to generate PDI interrupt.
  - Sync0 interrupt is enabled if DC Synchronous
  - AL Event are polled for events every ESC Read/Write to be handled by the stack
  - ecat_slv_poll() get called regular to handle stack operations
  - DIG_process(DIG_PROCESS_WD_FLAG) get called regular to kick watchdog counter
- Interrupt (SM or DC Synchronous)
  - PDI interrupt is enabled
  - SMCHANGE, EEP, CONTROL, SM0 and SM1 masked to generate PDI interrupt.
  - Sync0 interrupt is enabled if DC Synchronous
  - AL Event are not polled for events
  - ecat_slv_isr() get called regular to handle stack operations
  - DIG_process(DIG_PROCESS_WD_FLAG) get called regular to kick watchdog counter

Structure configuration parameter

- User Input - Optional
- Stack parameter configuration - Mandatory

- Stack and Application interaction functions - Optional and Mandatory depending on mode of operations

```
static esc_cfg_t config =
{
   /* User input to stack */
   .user_arg = NULL, /* passed along to ESC_config and ESC_init */

   /* Manadatory input to stack */
   .use_interrupt = 1, /* flag telling the stack the user application will use
                          interrupts, 0= Polling, 1 = Mixed Polling/Interrupt
                          and Interrupt*/

   .watchdog_cnt = 100, /* non UNIT watchdog counter, for the application
                           developer to decide UINT. This example set 100
                           cnt and by calling ecat_slv or
                           DIG_process(DIG_PROCESS_WD_FLAG) every 1ms,
                           it creates a watchdog running at ~100ms. */

   /* Values taken from config.h to configure the stack mailboxes and SyncManagers */
   .mbxsize = MBXSIZE,
   .mbxsizeboot = MBXSIZEBOOT,
   .mbxbuffers = MBXBUFFERS,
   .mb[0] = {MBX0_sma, MBX0_sml, MBX0_sme, MBX0_smc, 0},
   .mb[1] = {MBX1_sma, MBX1_sml, MBX1_sme, MBX1_smc, 0},
   .mb_boot[0] = {MBX0_sma_b, MBX0_sml_b, MBX0_sme_b, MBX0_smc_b, 0},
   .mb_boot[1] = {MBX1_sma_b, MBX1_sml_b, MBX1_sme_b, MBX1_smc_b, 0},
   .pdosm[0] = {SM2_sma, 0, 0, SM2_smc, SM2_act},
   .pdosm[1] = {SM3_sma, 0, 0, SM3_smc, SM3_act},

   /* Optional input to stack for user application interaction with the stack
    * all functions given must be implemented in the application.
    */
   .pre_state_change_hook = NULL, /* hook called before state transition */
   .post_state_change_hook = NULL, /* hook called after state transition */

   .application_hook = NULL, /* hook in application loop called when
                                DIG_process(DIG_PROCESS_APP_HOOK_FLAG) */
   .safeoutput_override = NULL, /* user override of default safeoutput when stack
                                   stop output */

   .pre_object_download_hook = NULL, /* hook called before object download,
                                        if hook return 0 the download will not
                                        take place */
   .post_object_download_hook = NULL, /* hook called after object download */

   .rxpdo_override = NULL, /* user override of default rxpdo */
   .txpdo_override = NULL, /* user override of default txpdo */

   /* Mandatory input to stack for SM and DC synchronous applications */
   .esc_hw_interrupt_enable = NULL, /* callback to function that enable IRQ
                                       based on the Event MASK */
   .esc_hw_interrupt_disable = NULL, /* callback to function that disable IRQ
                                        based on the Event MASK */

   /* Mandatory input for emulated eeprom */
   .esc_hw_eep_handler = NULL /* callback to function that handle an emulated eeprom */
};
```

The stack is setup but not running.

```
void main_run(void * arg)
{
...
   ecat_slv_init(&config);
```

4 EtherCAT slave stack API

4.1 DIG_process - Processdata handler

Implements the watch-dog counter to count if the slave should make a state to change SAFEOP due to missing incoming SM2 events. Updates local I/O and run the application in the following order, call read EtherCAT outputs, execute user provided application hook and call write EtherCAT inputs.

- #define DIG_PROCESS_INPUTS_FLAG 0x01
- #define DIG_PROCESS_OUTPUTS_FLAG 0x02
- #define DIG_PROCESS_WD_FLAG 0x04
- #define DIG_PROCESS_APP_HOOK_FLAG 0x08

Parameters

in	*flags*	= User input what to execute

```
void DIG_process (uint8_t flags);
```

4.2 ecat_slv_isr - Non-synchronous interrupt handler

ISR for SM0/1, EEPROM and AL CONTROL events in a SM/DC synchronization application

```
void ecat_slv_isr (void);
```

4.3 ecat_slv_poll - Interrupt polling routine for non-synchronous interrupt

Poll SM0/1, EEPROM and AL CONTROL events in a SM/DC synchronization application

```
void ecat_slv_poll (void);
```

4.4 ecat_slv_init - Stack initialization

Poll all events in a free-run application

```
void ecat_slv (void);
```

4.5 ecat_slv_init - Stack initialization

Parameters

in	*config*	= User input how to configure the stack

```
void ecat_slv_init (esc_cfg_t * config);
```

5 EtherCAT HW Layer implementations

5.1 EtherCAT HW Layer Enable/Disable interrupts

When running in polling or mixed polling/interrupt mode AL Event must be polled. Best done in ESC_read and ESC_write

```
void ESC_read (uint16_t address, void *buf, uint16_t len)
{
   ESCvar.ALevent = <HW read ALevent>;
   memcpy (buf, ESCADDR(address), len);
}

void ESC_write (uint16_t address, void *buf, uint16_t len)
{
   ESCvar.ALevent = <HW read ALevent>;
   memcpy (ESCADDR(address), buf, len);
}
```

5.2 EtherCAT HW Layer Enable/Disable interrupts

When running in mixed polling/interrupt or interrupt mode the EtherCAT HW Layer should provide with function to enable and disable interrupts, those functions will be called by the EtherCAT Slave Stack with mask to enable/disable SM2 and DC events, DC if DC us active. The code below acts as Pseudo code since access to ESC registers varies.

Example SM- or DC synchrounous, only PDI interrupt is used

```
void ESC_interrupt_enable (uint32_t mask)
{
   AL_EVENT_MASK = mask;
}

void ESC_interrupt_disable (uint32_t mask)
{
   AL_EVENT_MASK &= ~mask;
}
```

Example DC synchrounous, PDI and separate sync0 is used.

```
void ESC_interrupt_enable (uint32_t mask)
{
   AL_EVENT_MASK = mask;
   if(mask & dc_mask)
   {
       int_enable(SYNC0);
   ...
}
void ESC_interrupt_disable (uint32_t mask)
{
   AL_EVENT_MASK &= ~mask;
   if(mask & dc_mask)
   {
       int_disable(SYNC0);
   ...
}
```

5.3 EtherCAT HW Layer SYNC0 interrupt

Sync0 used in DC synchronous is generated from a ESC internal blocking for Distributed Clocks, sync0 interrupts can be setup as a se1perate interrupt source or part of the PDI interrupt. In most cases it is

used together with SM2 interrupts, therefor DIG_process allow to flag what parts it would execute. In this example sync0 is expected to happen after PDI isr SM2, SM2 handle DIG_process(DIG_PROCESS_OUTPUTS_FLAG) copying the RxPDO to local variables. Sync0 finish with executing the application and writing the TxPDO for next frame, DIG_process(DIG_PROCESS_APP_HOOK_FLAG | DIG_PROCESS_INPUTS_FLAG).

Example DC synchronous with separate sync0 interrupt

```
void sync0_isr (void * arg)
{
    DIG_process(DIG_PROCESS_APP_HOOK_FLAG | DIG_PROCESS_INPUTS_FLAG);
}
\encode

\subsection soesstack_eschw_interrupt_pdi EtherCAT HW Layer PDI interrupt

PDI interrupt is a collection of interrupt sources, consult the ESC reference manual
for details. Assume sync0 is handled by a separate interrupt and SM2 is mandatory for any interrupt
mode, depending on software support the application can choose to poll
AL event for non-synchronous interrupts running mixed polling/interrupt mode Or run
interrupt mode where no polling of AL event is needed. The code below acts as Pseudo code
since access to ESC registers varies.

Example SM- or DC Synchronous running mixed polling/interrupt mode

\code
void pdi_isr (void * arg)
{
   /* High prio interrupt used for synchronization */
   if(ESCvar.ALevent & ESCREG_ALEVENT_SM2)
   {
      /* If DC sync is not active, run the application, all except for the Watchdog */
      if(ESCvar.dcsync == 0)
      {
         DIG_process(DIG_PROCESS_OUTPUTS_FLAG | DIG_PROCESS_APP_HOOK_FLAG |
               DIG_PROCESS_INPUTS_FLAG);
      }
      /* If DC sync is active, call output handler only */
      else
      {
         DIG_process(DIG_PROCESS_OUTPUTS_FLAG);
      }
   }
}
```

Example SM- or DC Synchronous running interrupt mode, add separate handler for non-synchronous interrupts to minimize execution time for interrupt. In this example a semaphore is used to start a low prio task.

```
void pdi_isr (void * arg)
{
...
   ESCvar.ALevent = AL_EVENT_MASK;
   /* High prio interrupt used for synchronization */
   if(ESCvar.ALevent & ESCREG_ALEVENT_SM2)
   {
      /* If DC sync is not active, run the application, all except for the Watchdog */
      if(ESCvar.dcsync == 0)
      {
         DIG_process(DIG_PROCESS_OUTPUTS_FLAG | DIG_PROCESS_APP_HOOK_FLAG |
               DIG_PROCESS_INPUTS_FLAG);
      }
      /* If DC sync is active, call output handler only */
      else
      {
         DIG_process(DIG_PROCESS_OUTPUTS_FLAG);
      }
   }
```

```
   /* Assume there is task support, handle low prio interrupts from background
    * task not blocking for coming SM2 interrupts
    */
   if(ESCvar.ALevent & (ESCREG_ALEVENT_CONTROL | ESCREG_ALEVENT_SMCHANGE
         | ESCREG_ALEVENT_SM0 | ESCREG_ALEVENT_SM1 | ESCREG_ALEVENT_EEP))
   {
      /* Mask interrupts while servicing them */
      AL_EVENT_MASK &= ~(ESCREG_ALEVENT_CONTROL | ESCREG_ALEVENT_SMCHANGE
            | ESCREG_ALEVENT_SM0 | ESCREG_ALEVENT_SM1 | ESCREG_ALEVENT_EEP);
      /* Signal background task */
      sem_signal(ecat_isr_sem);
   }
}
```

5.4 EtherCAT HW Layer initialization

The ESC hardware layer is possible to place in the SOES library or Application project, to handle both scenarios the EtherCAT Stack pass the ESC config as parameter to ESC_init. ESC config has an user_arg that can be used for miscellaneous information passed from the application to ESC hardware initialization. ESC_init should setup the hardware to serve the application with necessary functions. The code below acts as Pseudo code since access to ESC registers varies.

Example SM- or DC Synchronous running mixed polling/interrupt mode

```
void ESC_init (const esc_cfg_t * config)
{
    /* Setup PDI interrupt */
    int_connect (IRQ_PDI, pdi_isr, NULL);
    int_enable (IRQ_PDI);

    /* Set mask to disable all interrupts, the stack enables SM2 interrupt */
    AL_EVENT_MASK = 0;

    /* Setup sync0 interrupt if DC synchronous */
    int_connect (IRQ_SYNC0, sync0_isr, NULL);
    /* Let the stack enable the DC interrupt */
    int_disable(IRQ_SYNC0);
}
```

Example SM- or DC Synchronous running interrupt mode

```
/* Non-synchronous interrupt handler function */
static void isr_run(void * arg)
{
   while(1)
   {
      sem_wait(ecat_isr_sem);
      ecat_slv_isr();
   }
}

void ESC_init (const esc_cfg_t * config)
{
    /* Create non-synchronous interrupt handler task and use a
     * semaphore for signaling
     */
    ecat_isr_sem = sem_create(0);
    task_spawn ("soes_isr", isr_run, 9, 2048, NULL);

    /* Setup PDI interrupt */
    int_connect (IRQ_PDI, pdi_isr, NULL);
    int_enable (IRQ_PDI);

    /* Set mask to enable non-synchronous interrupts to be able to operate,
     * let the stack enable SM2 interrupt.
     */
```

```
    AL_EVENT_MASK = (ESCREG_ALEVENT_SMCHANGE |
                     ESCREG_ALEVENT_EEP |
                     ESCREG_ALEVENT_CONTROL |
                     ESCREG_ALEVENT_SM0 |
                     ESCREG_ALEVENT_SM1);

    /* Setup sync0 interrupt if DC synchronous */
    int_connect (IRQ_SYNC0, sync0_isr, NULL);
    /* Let the stack enable the DC interrupt */
    int_disable(IRQ_SYNC0);
}
```

6 Run the application

For examples and references goto <SOES>/applications and <SOES>/soes/hal.

6.1 EtherCAT Slave Stack Polling

Configure not to use interrupts, call the EtherCAT stack handler periodically. This correspond to legacy use of SOES, soes_init , while(1) { soes(); }.

```
static esc_cfg_t config =
{
..
   .use_interrupt = 0,
..
};
void main_run(void * arg)
{
    ecat_slv_init(&config);
    while(1)
    {
        ecat_slv();
    }
}
```

6.2 EtherCAT Slave Stack Mixed Polling/Interrupt Mode

Configure to use interrupts, set interrupt enable/disable handlers, call the EtherCAT slave stack handler for polling non-synchronous interrupts periodically. Kick the watchdog supervising for incoming data from the EtherCAT Master

```
static esc_cfg_t config =
{
..
   .use_interrupt = 1,
   .esc_hw_interrupt_enable = ESC_interrupt_enable,
   .esc_hw_interrupt_disable = ESC_interrupt_disable,
..
};
void main_run(void * arg)
{
    ecat_slv_init(&config);
    while(1)
    {
        /* Kick watchdog with watchdog count intervals */
        DIG_process(DIG_PROCESS_WD_FLAG);
        ecat_slv_poll();
    }
}
```

6.3 EtherCAT slave stack interrupt mode

Configure to use interrupts, set interrupt enable/disable handlers, kick the watchdog supervising for incoming data from the EtherCAT Master

```
static esc_cfg_t config =
{
..
   .use_interrupt = 1,
   .esc_hw_interrupt_enable = ESC_interrupt_enable,
   .esc_hw_interrupt_disable = ESC_interrupt_disable,
..
};
void main_run(void * arg)
{
    ecat_slv_init(&config);
    while(1)
    {
        /* Kick watchdog with watchdog count intervals */
        DIG_process(DIG_PROCESS_WD_FLAG);
        task_delay(1)
    }
}
```