acontis technologies GmbH

SOFTWARE

EC-Master

Feature Pack Split-Frame-Processing

Version 3.2

Edition: July 23, 2024

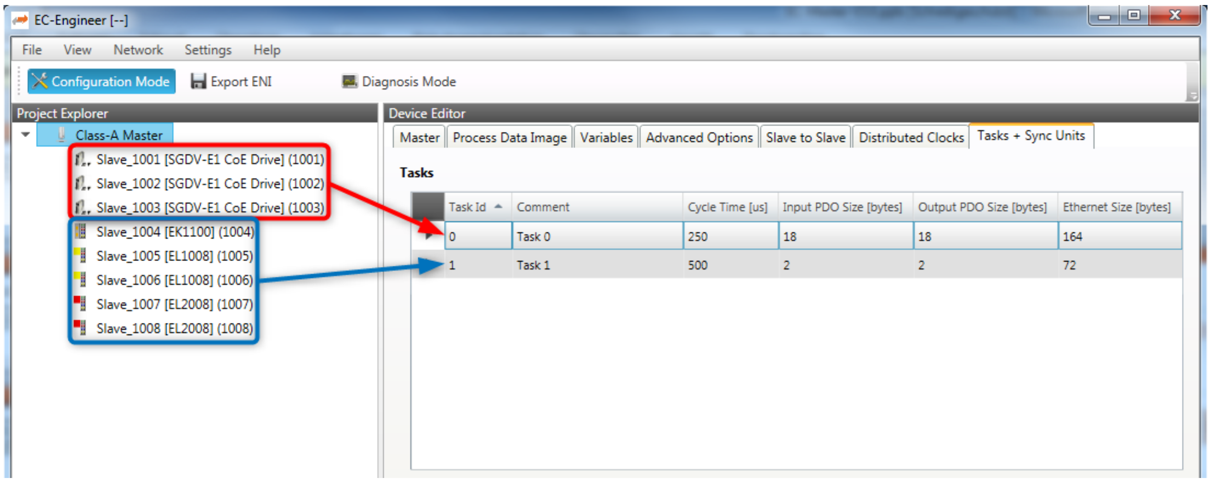
Contents

1 Introduction

The Split-Frame Processing Feature Pack enables the processing of multiple EtherCAT cyclic tasks in separate application threads. From the application perspective, this makes it possible to structure EtherCAT process data across multiple threads. Therefore, process data that requires different cycle times can be handled accordingly. Additionally, the processing of acyclic communication can also be outsourced to a separate thread.

For example, a typical application for the Split-Frame Processing Feature Pack would have one tasks for servo drives that requires very fast cycle times, and another task for I/O data that does not need very fast processing or special cycle time requirements.

EC-Engineer makes it easy to create configurations with several EtherCAT cyclic tasks:


EC-Engineer [--]
File View Network Settings Help
Configuration Mode
Export ENI
Diagnosis Mode
Project Explorer
Device Editor
Master Process Data Image Variables Advanced Options Slave to Slave Distributed Clocks Tasks + Sync Units

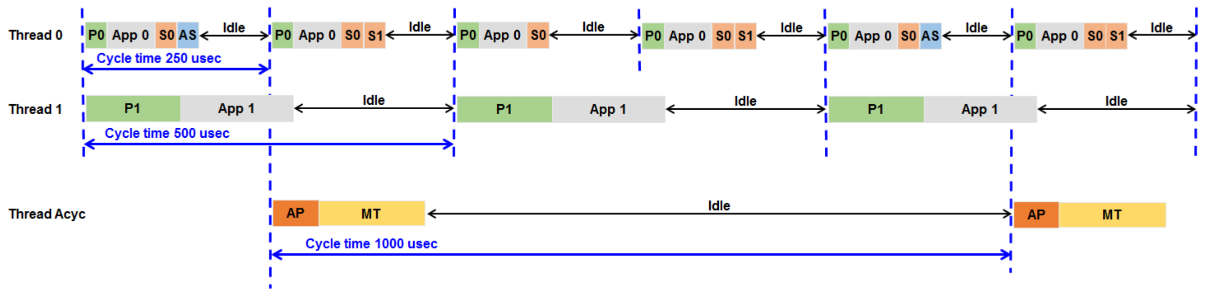
Class-A Master
- Slave_1001 [SGDV-E1 CoE Drive] (1001)
- Slave_1002 [SGDV-E1 CoE Drive] (1002)
- Slave_1003 [SGDV-E1 CoE Drive] (1003)
- Slave_1004 [EK1100] (1004)
- Slave_1005 [EL1008] (1005)
- Slave_1006 [EL1008] (1006)
- Slave_1007 [EL2008] (1007)
- Slave_1008 [EL2008] (1008)

Tasks

Task Id	Comment	Cycle Time [us]	Input PDO Size [bytes]	Output PDO Size [bytes]	Ethernet Size [bytes]
0	Task 0	250	18	18	164
1	Task 1	500	2	2	72

The timing of an application utilizing Split-Frame Processing application would look like the following:

- Thread 0 runs with a 250 microsecond cycle time and handles the processing of EtherCAT task 0. This thread also sends out the process data for all other tasks.
- Thread 1 runs with a 500 microsecond cycle time and handles only the processing of EtherCAT task 1.
- Thread Acyc runs with a 1 millisecond cycle time and processes only the acyclic communication and also the Master management tasks.



The operation modes polling and interrupt of the Ethernet Driver are supported.

2 Application programming interface, reference

In interrupt mode, all incoming frames are buffered. If all frames of a specific cyclic task have been buffered, the application will be informed by the CYCFRAME_RX_CB mechanism

See also:

emIoControl - EC_IOCTL_REGISTER_CYCFRAME_RX_CB in EC-Master ClassB manual.

2.1 emIoControl - EC_IOCTL_SET_SPLIT_FRAME_PROCESSING_ENABLED

Enable or disable the split frame processing. Default: Disabled.

If split frame processing is enabled the master allocates several buffers to store cyclic and acyclic EtherCAT frames. The size of these buffers depends on the number of cyclic frames in the ENI and the parameter `EC_T_INIT_MASTER_PARMS::dwMaxAcycFramesQueued` configured in `emInitMaster()`.

The functionality should be enabled between `emInitMaster()` and the start of the job task.

emIoControl - EC_IOCTL_SET_SPLIT_FRAME_PROCESSING_ENABLED

Parameter

- `pbyInBuf`: [in] Pointer to value of EC_T_BOOL. EC_TRUE: enable, EC_FALSE: disable.
- `dwInBufSize`: [in] Size of the input buffer provided at pbyInBuf in bytes.
- `pbyOutBuf`: [out] Should be set to EC_NULL
- `dwOutBufSize`: [in] Should be set to 0
- `pdwNumOutData`: [out] Should be set to EC_NULL

Return

EC_E_NOERROR or error code

2.2 emExecJob

```
EC_T_DWORD emExecJob(
    EC_T_DWORD dwInstanceID,
    EC_T_USER_JOB eUserJob,
    EC_T_USER_JOB_PARMS *pUserJobParms
)
```

Execute or initiate the requested master job.

To achieve maximum speed, this function is implemented non re-entrant. It is highly recommended that only one single task is calling all required jobs to run the stack. If multiple tasks are calling this function, the calls have to be synchronized externally. Calling it in a context that doesn't support operating system calls can lead to unpredictable behavior.

Parameters

- **`dwInstanceID`** – [in] Instance ID (Multiple EtherCAT Network Support)
- **`eUserJob`** – [in] user requested job
- **`pUserJobParms`** – [in] optional user job parameters

Returns

- EC_E_NOERROR if successful
- EC_E_INVALIDSTATE if master isn't initialized
- EC_E_INVALIDPARM if dwInstanceID is out of range or the output pointer is EC_NULL
- EC_E_LINK_DISCONNECTED if the link is disconnected
- EC_E_FEATURE_DISABLED for eUsrJob_SwitchEoeFrames if EC_IOCTL_SET_EOE_DEFFERED_SWITCHING_ENABLED hasn't be called before
- EC_E_ADS_IS_RUNNING if ADS server is running

Additionally to the standard jobs, the following eUserJob are defined for split frame processing:

enumerator **`eUsrJob_ProcessRxFramesByTaskId`**
Receive frames and process received data related to a specific task id (ENI: Cyclic/TaskId)

enumerator **`eUsrJob_ProcessAcycRxFrames`**
Receive frames and process received data related to acyclic frames

enumerator **`eUsrJob_SendCycFramesByTaskId`**
Send cyclic frames related to a specific task id (ENI: Cyclic/TaskId)

1. *`eUsrJob_ProcessRxFramesByTaskId`*
   When split frame processing is enabled, this call will process all currently received frames related to the specified task. This job can be called if the Ethernet Driver is configured in polling or in interrupt mode.

   struct `EC_T_USER_JOB_PARMS::`**`_PROCESS_RXFRAME_BY_TASKID`**

   ### Public Members

   EC_T_BOOL **`bCycFramesProcessed`**
   [out] Indicates whether all previously initiated cyclic frames of a specific cyclic task are received and processed

   EC_T_DWORD **`dwTaskId`**
   [in] Task ID of the cycle whose frames are to be processed (ENI: Cyclic/TaskId)

2. *`eUsrJob_ProcessAcycRxFrames`*
   When split frame processing is enabled, this call will process all currently received acyclic frames. This job can be called if the Ethernet Driver is configured in polling or in interrupt mode.

3. *`eUsrJob_SendCycFramesByTaskId`*
   Send cyclic frames related to a specific task id. If more than one cyclic entries are configured this user job can be used to send the appropriate cyclic frames. All frames stored in cyclic entries with the given task id will be sent.

   struct `EC_T_USER_JOB_PARMS::`**`_SEND_CYCFRAME_BY_TASKID`**

Public Members

EC_T_DWORD **`dwTaskId`**

[in] Task ID of the cycle whose frames are to be sent (ENI: Cyclic/TaskId)

3 Example code

3.1 Configuration

Enable split frame processing.

```
dwRes = emInitMaster(dwInstanceId, &oInitMasterParms);
/* Enable split frame processing */
{
    EC_T_BOOL bEnableSplitFrameProc = EC_TRUE;
    EC_T_IOCTLPARMS oIoCtlParms;

    OsMemset(&oIoCtlParms, 0, sizeof(EC_T_IOCTLPARMS));
    oIoCtlParms.pbyInBuf = (EC_T_BYTE*)&bEnableSplitFrameProc;
    oIoCtlParms.dwInBufSize = sizeof(EC_T_BOOL);

    dwRes = emIoControl(dwInstanceId, EC_IOCTL_SET_SPLIT_FRAME_PROCESSING_ENABLED,␣
↪&oIoCtlParms);
}
/* create cyclic task to trigger jobs */
```

Only in interrupt mode! Register RX callback function (see also EcMasterDemoSyncSm)

```
{
    EC_T_CYCFRAME_RX_CBDESC oCyRxDesc;
    EC_T_IOCTLPARMS         oIoCtlParms;

    /* setup callback function which is called after RX */
    OsMemset(&oCyRxDesc, 0, sizeof(EC_T_CYCFRAME_RX_CBDESC));
    oCyRxDesc.pfnCallback = CycFrameReceivedCallback;
    oCyRxDesc.pCallbackContext = EC_NULL;

    OsMemset(&oIoCtlParms, 0, sizeof(EC_T_IOCTLPARMS));
    oIoCtlParms.dwInBufSize = sizeof(oCyRxDesc);
    oIoCtlParms.pbyInBuf = (EC_T_PBYTE)&oCyRxDesc;

    dwRes = emIoControl(dwInstanceId, EC_IOCTL_REGISTER_CYCFRAME_RX_CB, &
↪oIoCtlParms);
}
```

3.2 Job Task

```
EC_T_USER_JOB_PARMS oJobParms;

/* process cyclic frames with task id 0 */
oJobParms.ProcessRxFramesByTaskId.dwTaskId = 0;
dwRes = emExecJob(dwInstanceId, eUsrJob_ProcessRxFramesByTaskId, &oJobParms);

/* process acyclic frames */
dwRes = emExecJob(dwInstanceId, eUsrJob_ProcessAcycRxFrames, EC_NULL);

/* myAppWorkpd(); */

/* send cyclic frames related to task id 0 */
oJobParms.SendCycFramesByTaskId.dwTaskId = 0;
dwRes = emExecJob(dwInstanceId, eUsrJob_SendCycFramesByTaskId, &oJobParms);

/* Execute some administrative jobs. No bus traffic is performed by this function␣
↪*/
dwRes = emExecJob(dwInstanceId, eUsrJob_MasterTimer, EC_NULL);

/* send queued acyclic EtherCAT frames */
dwRes = emExecJob(dwInstanceId, eUsrJob_SendAcycFrames, EC_NULL);
```

See also:

EcMasterDemo for polling or EcMasterDemoSyncSm for interrupt mode.