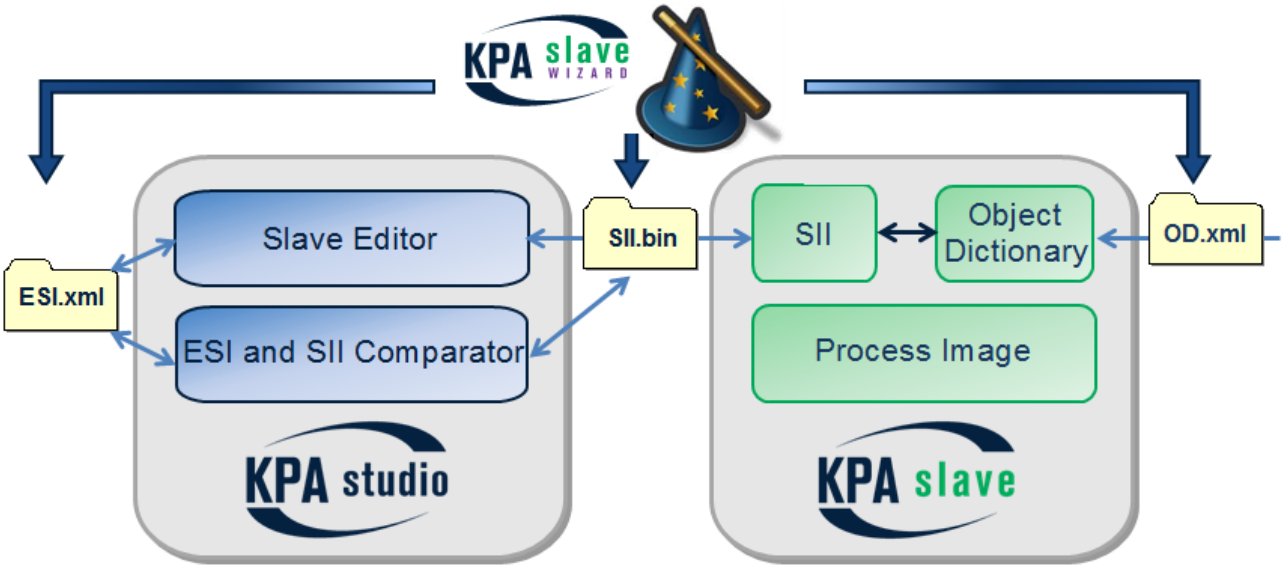
Why KPA EtherCAT® Slave Development Kit?

Date: 2015/01/12

KPA EtherCAT Slave Development Kit is a complete tool chain containing the KPA Slave Stack itself, KPA EtherCAT Studio with the KPA Slave Wizard and KPA Master for Windows.
The software stack is designed to run on microcontrollers, CPUs or DSPs.
Additionally KPA offers for one year assistance in conformance testing, technical support and update.



1. Current State

a) At present Slave Stacks are offered poor API and without deeper support. A flat learning curve is result.
b) The Object Dictionary requires several iterations. With each ESI and SII are to be adapted as well. Manual synchronization is necessary.
c) Sometime stack needs an operating system. A system abstraction layer is not provided.

2. Requirements

Powerful API; one tool keeping all files consistent and an abstraction layer.

3. KPA Solution

a) **KPA Slave Wizard** creates data structures for PDOs and Object dictionary according to any profile.
b) **SII Editor** is a development-tool in KPA EtherCAT Studio for editing slave's SII and saving consistently as ESI. Reading slave´s info from ESI, binary file or from slave and saving SII data to binary file or load to slave. **ESI and SII Comparator** detects differences to compare SII content acquired directly from a slave or provided in a file and highlighted them with displayed customizable colors.
c) Slave Stack already adapted to hardware platforms (x86, PPC, ARM) and certain operating systems like Xenomai for User/Kernel space

4. Delivery

KPA Slave DK Basic with CoE, FoE
KPA Slave DK Standard with EoE, SoE, VoE and dynamic OD generation, additionally to Basic
Both include:
+ Stack in source code
+ one KPA Studio Premium with Slave Wizard (release Q2/2014) and KPA Windows Master
+ support and maintenance for one year

5. Comparison between KPA's Slave Stack and others

Feature	KPA Slave Stack	Others
Number of supported input/output SMs	Not limited in the stack	Only one input and one output SM are supported in the samples
Process Images processing	API functions for data exchange between Process Images and objects (usually call of one function is enough)	The Process Images processing must be implemented in the application
Object Dictionary creation	Creation in application code or loading from an OD.xml-file (standard format)	Creation in application code
Binding objects with variables	Each object's entry can be bound with individual variable (by its pointer)	An object can't consist of some different variables – only one pointer can be bound to the whole of the object (e.g. a structure with all variables for the object's entries)
CoE object parameters	All parameters are supported, the processing is implemented in the stack	MIN, MAX, DEFAUL values for objects are not supported
Mapping flexibility	No difference for user between fixing and not fixing mapping	Only fixing mapping is implemented in the sample applications, implementations are applications-specific
CoE SDO processing	Full support in the stack	Mailbox fragmentation is not supported for objects and entries description
Slave stack usage simplicity	For general slave application it is enough to call a little number of API functions	Some slave functionality must be implemented in the application (e.g. Process Image processing)
SDO and state-change events	Callbacks for SDO and state change events are supported and can be set/removed at runtime	Code for SDO and state change events processing must be included in the slave stack sources
Slave stack library	Operating System Abstraction Layer (OSAL) using a list of functions – a static library can be created for platform independent part of the stack	Adaptation to platform using definitions (#define, #ifdef,…) - platform-independent stack can't be implemented as library; No OSAL