This work-in-progress paper was presented as part of the main technical program at IEEE ETFA'2011

Increasing EtherCAT Performance Using Frame Size Optimization Algorithm

Mladen Knezic, Branko Dokic, and Zeljko Ivanovic
University of Banja Luka, Faculty of Electrical Engineering
Patre 5, 78000 Banja Luka, Bosnia and Herzegovina
{mladen.knezic, bdokic, zeljko.ivanovic}@etfbl.net

Abstract

EtherCAT protocol is a popular Real-Time Ethernet solution that offers the highest communication efficiency in a number of operating conditions. However, for huge networks with several hundreds of devices distributed in the field, EtherCAT performance can become a critical factor.

In this paper, we propose a solution for increasing EtherCAT performance using a frame size optimization algorithm. The optimization algorithm is described and analyzed for a simple scenario present in typical automation systems.

1. Introduction

EtherCAT is one of the leading Real-Time Ethernet (RTE) solutions available on the market. Because of its communication mechanism that relies on the summation frame principle and the efficient addressing scheme (called logical addressing), this protocol offers the highest communication efficiency in a number of operating conditions. It is particularly suitable for networks with a large number of devices that have a small payload.

Despite its high communication efficiency, EtherCAT performance can become critical factor when used in huge networks that consist of several hundreds of distributed field devices (sensors and actuators). Since EtherCAT inherently uses the logical ring topology, network propagation time is greatly influenced by the number of networked devices. Given that EtherCAT uses summation frame for communication, frame transmission time also depends on the number of networked devices.

There are several approaches that can be used to increase EtherCAT performance. One of the approaches is to increase the bit rate to gigabit Ethernet. In that way, frame transmission time is expected to be decreased by the order of magnitude. Pros and cons of this approach are elaborated in more detail in [1] and [2]. The approach proposed in [3] and [4] is based on reducing the network propagation time using the parallelization technique called Switched EtherCAT. In that way, one large logical ring is broken into several rings with smaller propagation delays. Frames sent by the master are distributed concurrently to all branches connected to the switch.

The approach we propose in this paper is based on the symmetric spatial distribution of the devices in network. Using the logical addressing and appropriate EtherCAT commands, frame transmission time can be reduced down to 50% as shown in [5].

The paper is organized as follows. In Section 2, the basics of the EtherCAT protocol are given. Section 3 covers the structure of the EtherCAT network description file. In Section 4, a proposed algorithm for frame size optimization is given and described. Section 5 shows one scenario of the applied algorithm. Finally, Section 7 discusses some of the main conclusions.

2. Basics of the EtherCAT protocol

EtherCAT protocol is a RTE solution developed by Beckhoff in 2003, which is used for efficient data transfer between distributed I/O devices in industrial communication systems. Also, it offers a high degree of synchronization among devices as shown in [6], which makes it a suitable solution for motion control applications.

Communication in EtherCAT relies on the summation frame principle, which enables very high bandwidth utilization. Using this technique, the process data of several devices are carried together within one or more Ethernet frames as opposed to protocols that use an individual frame approach in which every frame carries process data for only one device. In order to differentiate devices, an EtherCAT telegram is used. Each telegram addresses one (or more) device in the network. Telegrams are encapsulated within standard Ethernet frames and sent by a device called the master. The master initiates all communication services in the system. EtherCAT frames travel across the network and return to the master across the same path after passing the last slave in logical ring. The EtherCAT frame structure is shown in Fig. 1.

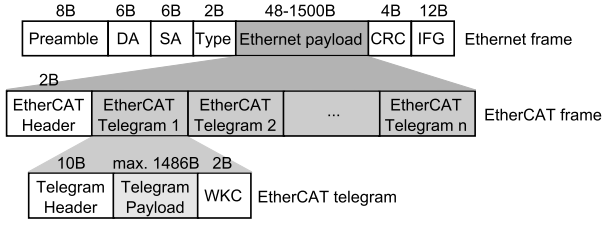

Figure 1. EtherCAT frame structure.

978-1-4577-0018-7/11/$26.00

EtherCAT currently supports two types of the physical layer: standard Ethernet 100BASE-TX and E-bus. The latter is intended for use in modular devices as a backbone communication bus and is fully compatible with the Ethernet standard.

2.1. Addressing modes

In EtherCAT, two addressing modes are supported: node (or device) and logical addressing. Node addressing is further divided into position (auto-increment) and fixed (configured station) addressing. Position addressing is typically used at the initialization phase of the network, while fixed addressing is used for online device configuration. For both types of addressing, the node address is defined by the ADP field in the EtherCAT telegram header (Fig. 1). The ADP field is a 16-bit value that specifies the position of the device in the ring (position addressing) or its configured address (fixed addressing). Along with the ADP, the ADO field is used for addressing the registers in the local memory of the slave device (64 KB address space).

Logical addressing, which refers to the location in the 4 GB logical process image shared by all networked devices, uses a 32-bit value (called ADR) in place of the ADP and ADO fields. The Fieldbus Memory Management Unit (FMMU) block in each slave is responsible for the translation between the logical and physical address (as illustrated in Fig. 2).

A special type of addressing mode is broadcast addressing in which all slaves are involved in the data exchange. This type of addressing is used for diagnostic messages (e.g. reading the state of all slaves in network).

2.2. Commands

A slave can be accessed using four basic communication services defined at the data-link layer: read, write, read/write, and read/multiple write [7].

Read access is used by the master to obtain input values from the slave registers, while write access enables writing to the slave output registers. Read/write access combines read and write services so that data can be exchanged in a bidirectional manner. Read/multiple write service reads the value from the addressed device (by using either position or fixed addressing) and writes it to the devices that follow. In EtherCAT, this service is typically used for the synchronization mechanism of the protocol.

Every service can be combined with any addressing mode, with the exception of read/multiple write service, which can be used only with the node addressing modes.

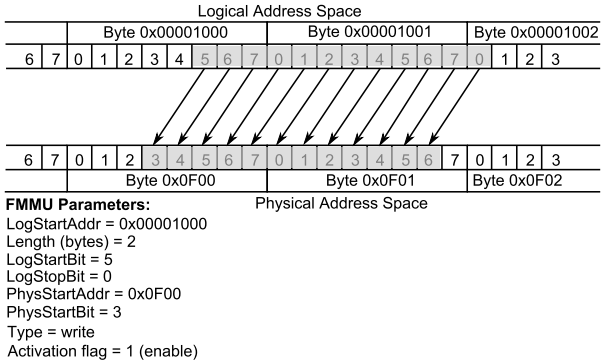

Figure 2. FMMU mapping example.

3. EtherCAT network description file

EtherCAT Network Information (ENI) is an XML-based file that contains the configuration settings of the EtherCAT network. It contains general information about the master and configurations of every slave device connected to the master (topology, position and configured slave address, process data size, etc.). It also defines commands that should be sent during the specific transitions in the EtherCAT State Machine (ESM). ENI file, created by the configuration tool, can be further used by the EtherCAT master driver [8].

Frame size optimization algorithm, described in Section 4, extracts necessary network information from ENI file created by the available configuration tools. In our examples, for ENI file creation, we have used two available configuration tools: Beckhoff TwinCAT System Manager [9] and KPA Studio EtherCAT [10].

4. Frame size optimization algorithm

In EtherCAT networks, frame size can be reduced by exploiting the symmetry of the devices. In a network with the equal number of inputs and outputs and symmetric spatial distribution of the devices, the size of the EtherCAT telegram required for the data exchange can be halved by using the LRW (logical read/write) command. For example, if EtherCAT module, which is connected to the master using an E-bus coupler (e.g. EK1100), contains 4 input terminals with 4 bits process data (e.g. EL1004) and 4 output terminals with 4 bits process data (e.g. EL2004), telegram size has to be equal to 4 bytes (telegram payload) when the input terminals are placed before the output terminals.

If we reorder slave terminals within the module so that the output terminals goes before the input terminals, the EtherCAT telegram size has to be only 2 bytes. This can be achieved by introducing the LRW command for data exchange. In this way, output data is first written to the output terminals. Since output data is not used in the remaining part of the network, we can use its space as a placeholder for inputs that have to be read from the input terminals. The idea is to arrange the slave devices in the network so that we first write outputs and then read inputs from the devices, or to alternately write/read data to/from devices. The pseudo code of the algorithm is shown in Fig. 3.

First, the initial value of the logical address for inputs ($LogAddrI$) and outputs ($LogAddrO$) must be set. Initially, these values are equal and usually set to zero. Then,

```
1: Set LogAddrI=LogAddrO initial value
2: for i=1 to n do
3:    Diff=LogAddrI-LogAddrO
4:    if Diff>Slave(i).Diff then
5:       Find i<k≤n for which Diff≤Slave(k).Diff
6:       if k was not found then
7:          Find i<k≤n for which min(|Slave(k).Diff|)
8:       end if
9:       Swap devices Slave(i) and Slave(k)
10:   end if
11:   if Slave(i).Obits≠0 then
12:      Calculate FMMU parameters for i^th slave
13:      if Diff>0 then
14:         LogAddrO=LogAddrI+Slave(i).Obits
15:      else
16:         LogAddrO=LogAddrO+Slave(i).Obits
17:      end if
18:   end if
19:   if Slave(i).Ibits≠0 then
20:      Calculate FMMU parameters for i^th slave
21:      LogAddrI=LogAddrI+Slave(i).Ibits
22:   end if
23: end for
```

Figure 3. Frame size optimization algorithm.

in each iteration, the difference between output and input logical addresses ($Diff$) must be calculated. This value is then compared to the difference in the slave device ($Slave(i).Diff$) that can be calculated using the equation $Slave(i).Obits - Slave(i).Ibits$, where $Ibits$ and $Obits$ are the size of the process data for inputs and outputs (in bits), respectively. If $Diff > Slave(i).Diff$, this means that the inputs do not fit into the outputs placeholder and outputs must be aligned with the logical address of the inputs from the previous iteration. This wastes bandwidth and reduces the communication efficiency. To overcome this problem, the algorithm scans the remaining part of the network in order to find the devices (at the k^{th} index) for which is $Diff \leq Slave(k).Diff$. If it finds such a device, it swaps it with the current (i^{th}) device, otherwise it takes the device with the smallest difference $|Slave(k).Diff|$ and swaps it with the current device.

The second part of the algorithm is responsible for calculating the $LogAddrI$ and $LogAddrO$ for the next device. The inputs and outputs logical addresses are simply calculated by adding the size of the slave input/output process data to the logical address value from the previous iteration (as shown in Fig. 3).

Device FMMU parameters can be calculated as

$$LogStartAddr = \lceil LogAddrX/8 \rceil \tag{1}$$

$$Length = \lceil Xbits/8 \rceil \tag{2}$$

$$\begin{aligned} LogStartBit =& LogAddrX - \\ & 8 \cdot (\lceil LogAddrX/8 \rceil - 1) \end{aligned} \tag{3}$$

$$\begin{aligned} LogStopBit =& NewLogAddrX - \\ & 8 \cdot (\lceil NewLogAddrX/8 \rceil - 1) - 1 \end{aligned} \tag{4}$$

where: X – is either I (inputs) or O (outputs), $NewLogAddrX$ – value of the $LogAddrX$ for the next iteration, $LogStartAddr$ – logical start address, $Length$ – length of the process data (in bytes), $LogStartBit$ – logical start bit, $LogStopBit$ – logical stop bit. Other FMMU parameters in the slave device (physical start address, physical start bit, type and activation flag) are not altered.

It is important to note that the algorithm is applied in a hierarchical manner, i.e. it is first applied to each of the EtherCAT modules, and then to the whole network. In this way, the position of the slaves that belong to the module is preserved (cannot be moved outside the module).

5. Results

The algorithm described in the previous section is applied to the relatively simple network whose topology is shown in Fig. 4. Details about devices used in the network are given in Table 1. We have applied the proposed

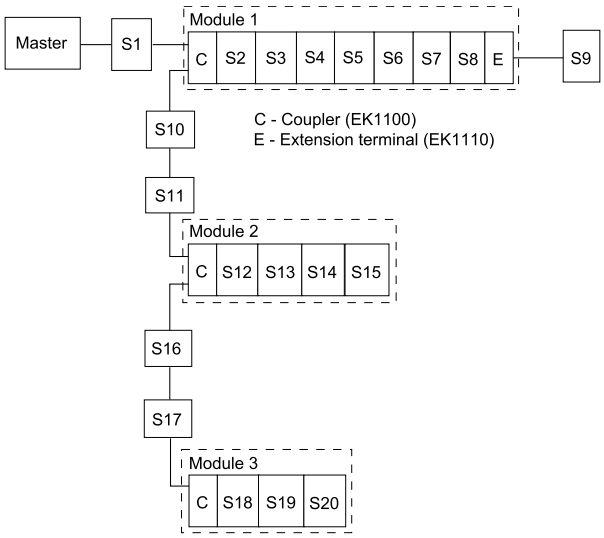

Figure 4. Example of the system.

Table 1. Slave device information.

Slave	Name	Ibits	Obits
S1	AX2000-B110	64	48
S2-S5, S12-S15	EL1004	4	-
S6-S8	EL2004	-	4
S9	EL9800-SPI	256	128
S10, S11	AX5101	48	48
S16	AX5203	96	96
S17	FB1111 Dig. In/Out	16	16
S18-S20	EL4034	-	64
Total:		560	588

algorithm for two scenarios. In the first scenario, topological changes are allowed for all devices in the network. Resultant topology is shown in Fig. 5. The second scenario allows topological changes only within modules. In this case, the optimization algorithm gives slightly worse results, but the main topology of the network is fully preserved (slave positions are modified only in Module 1 as shown in Fig. 5). The results of the optimization algorithm are compared with the results obtained using available configuration tools. Details about this comparison are given in Table 2.

From this table we can see that in comparison to TwinCAT configuration tool, frame transmission time is about 32% shorter when topological changes are allowed and about 21% when topological changes are allowed only within modules. When KPA Studio EtherCAT is used, results become even better (over 45% shorter frame transmission time for both cases).

6. Conclusion

In this paper, a frame size optimization algorithm that can be used for increasing EtherCAT performance has been proposed. It exploits the symmetric spatial distribution property of the EtherCAT networks in which input/output data can be exchanged more efficiently using the LRW command (one EtherCAT telegram). This solution is inexpensive and flexible so it can be easily implemented in software. Moreover, other functionalities of the EtherCAT network, e.g. redundancy, synchronization, hot-plugging etc., are not jeopardized by this solution.

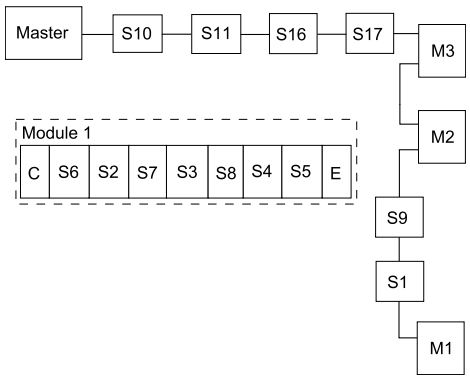

Figure 5. Network topology obtained using proposed algorithm.

Table 2. Performance comparison.

Configuration tool	L_{Frame} (bytes)	T_{Frame} (µs)
Algorithm applied on whole network	126	10.08
Algorithm applied on modules only	146	11.68
TwinCAT System Manager	185	14.80
KPA Studio EtheCAT	273	21.84

Results obtained for a relatively simple EtherCAT network structure are promising. In more complex setups, it is expected that the algorithm will provide even greater benefits. Although we have used an offline configuration of the network, the algorithm can be easily adapted for online configuration.

However, this algorithm has some drawbacks too. In many applications, network topology is dictated by the environment constraints and nature of the application. In these cases, it is not allowed to change the network topology. This difficulty can be overcome by using the algorithm without slave reordering (except within modules since the slaves within modules are physically close to each other and can be swapped easily). Also, there are cases when symmetry cannot be achieved (e.g. networks that contain only inputs). In such cases, different solutions must be used.

Additionally, network diagnostic is somewhat worsened because the data exchange is carried out using a single EtherCAT telegram. The most common way of improving network diagnostic in EtherCAT is to divide the network into logical groups of devices (that perform similar functions) called SyncUnits (or domains). By combining the concept of SyncUnits with the proposed algorithm, some trade-off could be achieved. This aspect will be more investigated in our future work.

References

[1] J. Jasperneite, M. Schumacher, and K. Weber, "Limits of increasing the performance of industrial Ethernet protocols," in *Proc. IEEE Int. Conf. Emerging Technologies and Factory Automation*, pp. 17–24, 2007.

[2] G. Prytz, "A performance analysis of EtherCAT and PROFINET IRT," in *Proc. IEEE Int. Conf. Emerging Technologies and Factory Automation*, pp. 408–415, 2008.

[3] G. Cena, S. Scanzio, A. Valenzano, and C. Zunino, "A Distribute-Merge Switch for EtherCAT Networks," in *Proc. 8th IEEE Int. Factory Communication Systems Workshop*, pp. 121–130, 2010.

[4] G. Cena, S. Scanzio, A. Valenzano, and C. Zunino, "Performance Analysis of Switched EtherCAT Networks," in *Proc. IEEE Conf. Emerging Technologies and Factory Automation*, pp. 1–4, 2010.

[5] M. Knezic, B. Dokic, and Z. Ivanovic, "Influence of the EtherCAT Frame Transmission Time on Network Efficiency in Case of Asymmetric Traffic," *Proc. 54th ETRAN Conf.*, Section EL4.1, pp. 1–4, 2010. (in Serbian)

[6] G. Cena, I. C. Bertolotti, S. Scanzio, A. Valenzano, and C. Zunino, "On the Accuracy of the Distributed Clock Mechanism in EtherCAT," in *Proc. 8th IEEE Int. Factory Communication Systems Workshop*, pp. 43–52, 2010.

[7] Beckhoff Automation GmbH, "Hardware Data Sheet ET1100 – EtherCAT Slave Controller, Ver. 1.8," 2010. [Online]. Available: http://www.beckhoff.com/

[8] EtherCAT Technology Group, "EtherCAT Network Information – Specification – ETG.2100, Ver. 1.0.0," 2009.

[9] Beckhoff Automation GmbH. TwinCAT System Manager. [Online]. Available: http://www.beckhoff.com/

[10] Koenig-PA GmbH. KPA Studio EtherCAT. [Online]. Available: https://koenig-pa.com