A Case Study on Multi-Softcore Aided Hardware Architectures for Powerline MAC-Layer

N. Rother, T. Stuckenberg, S. Nolting, C. Uhlemann and H. Blume
Institute of Microelectronic Systems
Leibniz Universität Hannover
Hanover, Germany
Email: {rother, stuckenberg, nolting, blume}@ims.uni-hannover.de

Abstract—**Powerline communication is a promising technology for connecting Internet of Things (IoT) applications, where devices have strict limitations regarding available installation space and power dissipation. Especially the wiring of these devices benefits from not having additional cables for network connection. Thus, saving costs and additional installation effort. In this paper a very resource-efficient implementation of a HomePlug 1.0.1 [5] compatible powerline MAC layer, which is used to control the data flow and link status of a powerline connection, is presented. The MAC layer is implemented in two variants, using state machines and softcore processors. A comparison of the two approaches shows that the softcore design used up to 78 % less FPGA ressources and is superior in terms of flexibility and maintainability.**

Keywords—FPGA, GCC, Multi-Core, Powerline, Softcore

I. Introduction

The ubiquitousness of smart, connected devices, often called "Internet of Things (IoT)", gained a lot of attraction in the last decade. These devices can be found in many different places like households, industry or public spaces. The interconnections of these devices is often realised with wireless technologies, such as WiFi, Bluetooth or ZigBee [2]–[4]. This allows, in conjunctions with a battery as the power supply, flexible placing of the devices. For applications like smart light bulbs, which are connected to the power grid, Powerline communication is another promising technology to realize the interconnection of different devices. The idea of using the existing power network for communication of domestic appliances has been proposed in [1].

For such devices small and energy-efficient powerline modems are required. Since 1990 various different powerline standard have emerged, among them HomePlug, HomePNA, Panasonic AV and the proposed unification IEEE 1901 [8]. This work will focus on the HomePlug standard, which is the dominant standard in Europe and Nothern America. The HomePlug Alliance has published four different standards, namely HomePlug 1.0 from 2001 [5], HomePlug AV from 2005 [8] and HomePlug AV2 from 2012 [6], as well as a reduced version of HomePlug AV for vehicle-to-grid communication, called HomePlug Green PHY in 2012 [7]. The later standards feature higher data rates, at the cost of increased complexity. As IoT applications typically do not need high data rates this work concentrates on the HomePlug 1.0.1 standard.

The HomePlug specification splits the system architecture in two layers, called Media Access Layer (MAC) and Physical Layer (PHY), in analogy to the well-known OSI model [17]. As shown in Fig. 1, the PHY layer handles the physical connection to the power line. It features OFDM (de)modulators and includes mechanisms for forward errror correction. The PHY layer is controlled completely by the MAC. This layer tracks the channel state and uses the PHY to send and receive data on the channel. The MAC layer connects to the Link Level Connection layer (LLC, not shown) to handle incoming and outgoing data. Other tasks of the MAC layer include the segmentation and reassembly of frames which are to large to be transmitted at once, and handling of collisions and other transmission errors. Furter details on the HomePlug protocol can be found in [9].

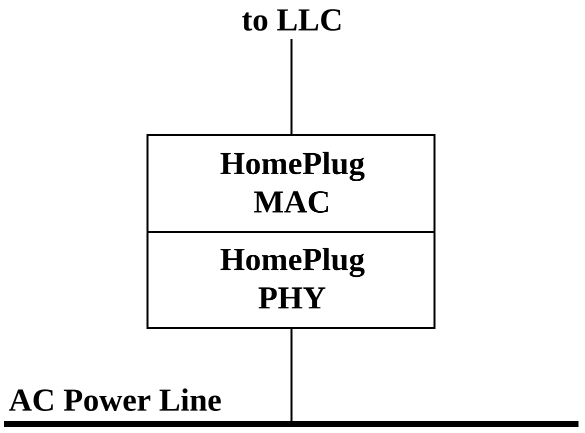

Fig. 1: HomePlug 1.0.1 layer architecture consisting of Link Level Connection (LLC), Media Access (MAC) and Physical Layer (PHY) [5].

In this work, a complete implementation of the HomePlug 1.0.1 MAC layer on an FPGA is presented. The PYH layer is not part of this work. Our goal was to evaluate, if using softcores processors could make the design more resource-efficient. The remainder of the this paper is organized as follows. Section II evaluates different softcore processors. Section III describes the HomePlug MAC layer, Section IV gives details of the implemented design. In Section V the designs are evaluated and Section VI gives a conclusion.

II. Softcore Processors

Softcore processors are microprocessors that are described in a hardware description language and can be synthesized

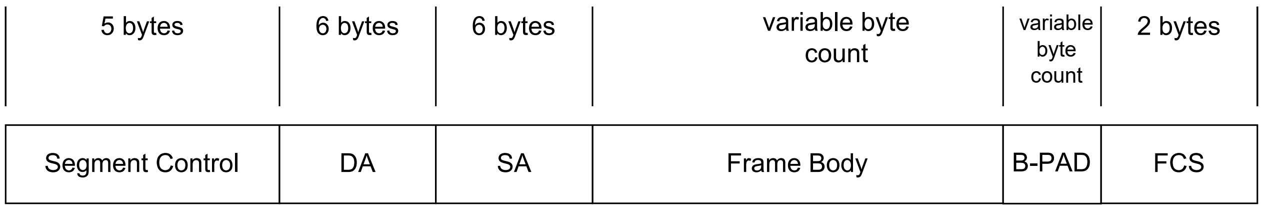

Fig. 2: HomePlug 1.0.1 segement structure [5].

TABLE I: Key characteristics of different softcores. Results for an Xilinx Virtex 6 FPGA with LUT-6.

	NEO430	openMSP	PauloBlaze	AVR_Core
Architecture	16 bit	16 bit	8 bit	8 bit
Extensibility	high	high	medium	low
LUT-6	849	1579	366	≈ 465[1]
Register	652	594	74	-

for an FPGA. They can then be programmed like a hardware processor ("hardcore") using regular programming languages like C. Their main advantage over dedicated hardware processors is the possibility to change or adapt the processor to suit the needs of the applications. The possible modifications include additional instructions, optimized register files and memories or hardware accelerators for special computations.

For this work four different softcore processors were evaluated for resource requirements and extensibility. The four candidates are NEO430, openMSP, PauloBlaze and AVR_Core [10]–[13]. Table I shows a summary of the evaluation. The NEO430 and the openMSP use a 16 bit architecture, the PauloBlaze and AVR_Core a 8 bit architecture.

The NEO430 [10] and the openMSP [12] are both re-implementations of the MSP430 instruction set architecture (ISA) from Texas Instruments. Due to this, the existing MSP430 compiler toolchain can be used to program these softcores in C or C++. The MSP430 ISA uses a 16-bit addressing scheme and features a rich instruction set that allows for compact code. All I/O devices are mapped into a single, unified memory space. Since only roughly 75 % of the available memory address space is used in the default configuration (depending on the size of the implemented instruction and data memory), there are enough free addresses in the address space for custom I/O extension. In both cores, all communication with peripherals is carried out though a bus system. This allows for easy extension with custom modules.

The MSP430 ISA is a mult-cycle architecture, i.e. every instruction takes multiple cycles to execute. There is no pipelining implemented. The openMSP instruction timing very much resembles the original, with every instruction taking 1–6 cycles. The NEO430 requires about 2 to 3 times more cycles for a single instruction due to a different implementation. On the other hand the NEO430 requires about half the FPGA resources compared to the openMSP (cf. Table I) and allows for higher clock frequencies. As our main focus was a resource-efficient implementation, the NEO430 was chosen for this work.

[1]optimistic estimation: 1860 LUT-4 divided by 4

The AVR_Core [13] and the PauloBlaze [12] are both 8-bit architectures which implement different instructions sets. The AVR_Core is compatible to the ATmega103 from Microchip/Atmel. There is a wide range of C and C++ compilers for this architecture. All I/O happens through a special 64-byte I/O area in the memory space. For the ATmega103 this area is completely filled with internal peripherals. Custom extensions would therefore require the removal of some of the default peripherals and are limited in the number of available I/O addresses.

The PauloBlaze is a re-implementation of the PicoBlaze ISA from Xilinx. This ISA is very limited and, to the best of our knowledge, there is no working C compiler for this architecture. All programming has to be done in assembler. Since the PauloBlaze implements the same instruction set as the PicoBlaze, the different available assemblers can be used without modification. The processors architecture includes up to 256 Byte of data memory and can address up to 4096 instructions. Connectivity to I/O devices is archived by 256 different byte-wide input and output ports. Every instruction of the PauloBlaze is executed in two cycles. Due to this the timing of the executed program is very deterministic. As the PauloBlaze has a higher extensibility and requires less resources (cf. Table I) than the AVR_Core it was chosen for this project.

III. HomePlug MAC Layer

The MAC layer in the HomePlug 1.0.1 standard is the interface between the LLC and PHY layer [5]. It prepares outgoing frames for transmission on the channel and handles retransmission in the event of transmission errors. Incoming frames are received, reassembled and acknowledged by the MAC layer. If a frame contains MAC management information, this data is processed by the MAC layer itself. All other payload data is forwarded to the LLC.

The HomePlug standard differentiates between *frames* and *segments*. Frames can be up to 1609 bytes long and typically encapsulate a complete Ethernet frame in the payload. They are constructed by the MAC layer with information from the LLC. For tranmission on the medium, a single frame is split up in multiple segments by the MAC layer. The size of a segment is specified by the number of transmitted OFDM symbols. A segment can contain up to 160 symbols. As HomePlug is using a variable modulation scheme based on channel quality, the number of bytes in an OFDM symbol is not fixed. As a consequence of this, the number of bytes in a segment can vary between 38 and 2076. Fig. 2 show the structure of a segment. The first 17 bytes contain header information, like the segment number and the MAC address of the source (SA)

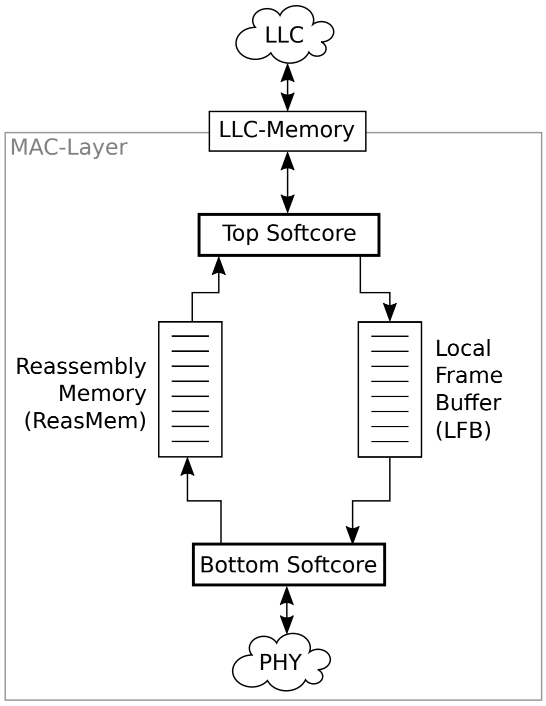

Fig. 3: Overview of the system architecture. The system consists of two softcores and interconnecting memories.

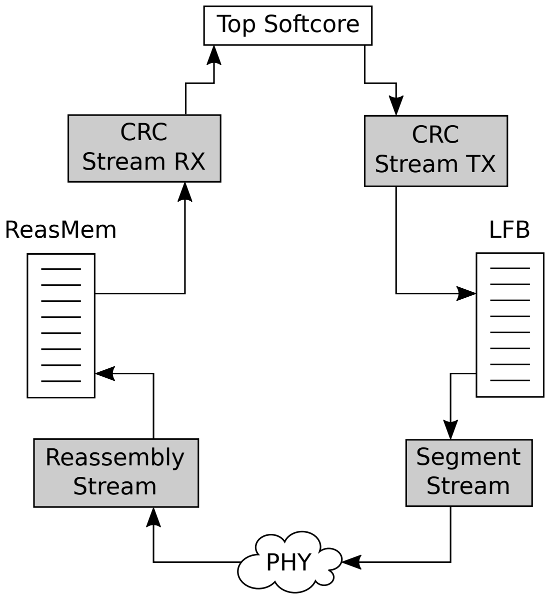

Fig. 4: Interconnection of the two softcores with the memories, using four “Streams”

and destination (DA). This header is followed by the payload, a portion of the frame to be segmented. For segmentation, the complete frame is treated as an uniform byte sequence. The remaining bytes contain padding bytes to fill up multiples of 20 OFDM symbols. The segment ends with two bytes containing a CRC16 checksum.

Another important task of the MAC layer is the tracking of the so-called “Virtual Carrier Sense”. For this, virtual states are assigned to the channel. The state tracking relies on synchronisation information called “FrameControl” as well as time keeping. Due to this channel occupation tracking, a collision on the channel is mostly avoided.

IV. Implementation

In this section, our design will be explained. The design subdivides in multiple softcore, hardware modules and memories. A general overview of the implementation is shown in Fig. 3. It can be seen that the system contains a total of two softcores, called “Top” and “Bottom”. Each softcore handles the communication with one interface of the MAC layer. The *Top Softcore* communicates with the LLC through the LLC memory. This link is not time critical, but includes a complex control flow. The *Bottom Softcore* connects to the PHY. As all time keeping on the channel is done by the MAC layer, this connection has real-time requirements. On the other hand, the control flow of this interface is very simple. The usage of two softcores in the system allows us to combine this conflicting requirements in a single system, while still using simple and therefore resource-efficient softcores.

The interconnection of the softcores is implemented using two memories for incoming and outgoing frames, called *Reassembly Memory* (ReasMem) and *Local Frame Buffer* (LFB), respectively. The general data flow is as follows. When the LLC provides new data to transmit, a MAC frame with header, payload, padding and checksum is generated by the *Top Softcore* and placed in the LFB. The *Bottom Softcore* is then notified that a new frame is ready for transmission. When the channel state tracking indicates that a transmission is now possible, the *Bottom Softcore* instructs the PHY to send out the frame, segment by segment. The necessary segment framing is generated on the fly (see below). The incoming data segments received from the PHY are place in the ReasMem by the *Bottom Softcore*. Special handling of memory addresses makes sure that the segments are written in a way that complete frames are reassembled in the ReasMem, even if the segments are received out of order or intermixed from different transmission (see below). When the last segment of a frame was received, the *Top Softcore* is notified, which will then read the frame from the ReasMem. All contained headers are evaluated and the payload is forwarded to the LLC.

As shown in Fig. 4 the ReasMem and LFB are not directly connected to the softcores, but through dedicated hardware modules, called “Streams”. These hardware modules present the underlying memory as a pseudo-FIFO interface to the softcores. During reading or writing, different operations are performed on-the-fly. These operations include encryption/decryption[2], multiplexing and calcuation of checksums.

As an example, a more detailed view of the *Segment Stream*, connecting the PHY to the LFB, is shown in Fig. 5. The Segment Stream consists of a four-way multiplexer and a small state-machine-based control unit that is connected to the *Top Softcore*. When a new segment is transmitted, the multiplexer selects the HeaderMem, containig the segment header.

[2]Encryption is optional in the HomePlug standard. Our current implementation does not contain support for encryption or decryption, but the streams included interfaces for this.

Subsequent reads by the PHY will now yield bytes from the segment header. After 17 bytes—the size of the header—the control unit automatically switches the multiplexer to the LFB. This occurs independent from the softcores. All following reads will now output bytes from the LFB in advance. The start and stop address in the LFB had to be configured by the *Top Softcore* to exactly include the part of the frame that is contained in the current segment. When the preconfigured number of bytes has been read from the LFB, the control unit switches the multiplexer to a source generating zero-bytes for padding. During the whole process, all read bytes are send to the CRC module, which generates a CRC16 checksum. At the end of the transmission, the multiplexer is switched to the CRC module to include the checksum of the transmitted data as the last two bytes of the segment.

Another architecture detail, the *Reassembly Stream*, is shown in Fig. 6. It connects the PHY to the ReasMem. Incoming data from the PHY is written to a FIFO. After 17 bytes—the segment header—have been received, the *Top Softcore* is notified via an interrupt. It reads out the header from the FIFO and computes the start address of the remaining bytes in the ReasMem. This address is send to a hardware write-back unit (shown as multiplexer in Fig. 6), which automatically flushes the FIFO to the ReasMem. The ReasMem address is chosen in a special way to automatically reassemble incoming frames in memory. This is shown in Fig. 7. The first segment is written to memory as-is. The header has already been read from FIFO, so only the payload, padding and CRC are written to memory. The second segment is slightly overlapping the first, overwriting the padding and checksum from the first. Using this technique, the payload can be later found in a continuous memory region.

The other three streams are implemented in a similar fashion. The *CRC Stream RX* and *CRC Stream TX*, connecting the *Top Softcore* to the LFB and ReasMem, encrypt/decrypt transmitted data and calculate the CRC32 checksum of it. The Reassembly Stream—connecting the PHY to the ReasMem—coordinates the writing of incoming segments to the ReasMem, so that whole frames are placed in order in the memory, even if the segments are received out of order.

V. Evaluation

Two version of the system described in Section IV were implemented. Both contain a NEO430 as the *Top Softcore*, but the *Bottom Softcore* was implemented with either a second NEO430 or a PauloBlaze. The whole design was synthesised using Vivado 2017.2 and then mapped to a Kintex 7 XC7K410T FPGA. Table II shows the mapping results. Additionally, as a baseline reference, a version without softcores was implemented. In that version, the whole MAC layer logic was implemented with state machines as given in the HomePlug standard.

As presented in Table II, the pure state-machine implementation requires 7845 LUTs and 5718 registers. The implementation with two NEO430 requires 1941 LUTs and 1401 registers, a significant reduction by 75 %. This can be explained by the fact that softcores consume a fixed amount of resources, regardless of the implemented function. Due to the complex control flow at the LLC interface, the hardware

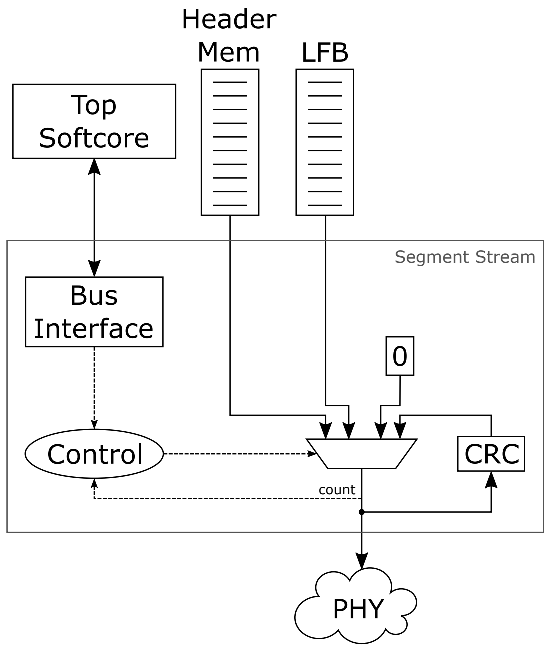

Fig. 5: Structure of the Segment Stream consisting of a multiplexer with a control unit connected to the *Top Softcore* and a CRC generator.

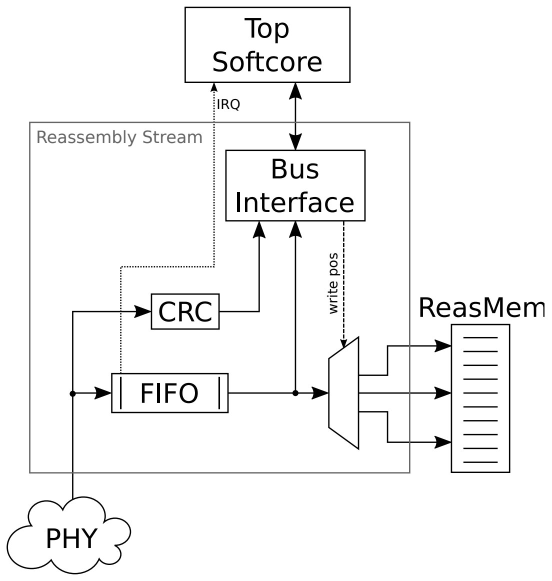

Fig. 6: Structure of the Reassembly Stream consisting of a FIFO, a CRC generator and a write-back unit (displayed as multiplexer).

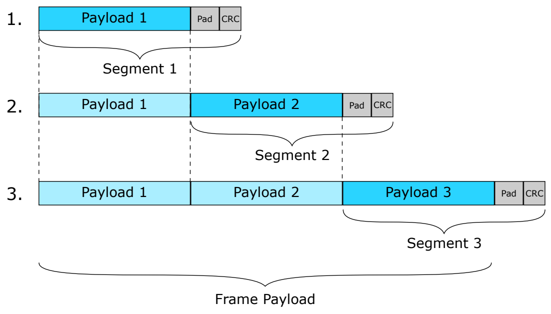

Fig. 7: Three segments beeing reassembled in memory. The payload is found in a continuous memory region due to partial overwriting of previous segments.

TABLE II: Key characteristics of the different implementations. Mapping was done using Vivado 2017.2, targeting a Xilinx Kintex XC7K410T FPGA

Variant	LUT-6	Register	BlockRAM
Reference	7845	5718	21
2xNEO	1941 −75 %	1401 −75 %	10,5 −50 %
1xNEO+1xPauloBlaze	1730 −78 %	1198 −79 %	9.5 −55 %

implementation of the corresponding state machine consists of a great amount of states, consuming large amounts of resources. Replacing the *Bottom Softcore* with a PauloBlaze further reduces the resource requirements by about 200 LUTs and registers. This implementation then requires 1730 LUTs and 1198 registers, 78 % less than the reference version. The amount of required BlockRAM is also reduced from 21 to 10.5 or 9.5, respectively. This is because the state machine based design does not implement the optimisation of the segmentation process described in Section IV, but stores every segment separatly. It should be noted that both softcore-based designs contain induvidual instruction and data memories for the softcores, which is not present in the state machine version. Our target clock frequency was 50 MHz, which is the sampling rate of the analog fronted of our HomePlug system. All design variants reach this target. Evaluation of the critical path showed that all three variants could even be run with a 100 MHz clock without violating timing constraints.

In a further analysis, the active time of the *Bottom Softcore* in the 26 µs long Response Interframe Space (RIFS) period was evaluated. This is the time after the end of a data transmission

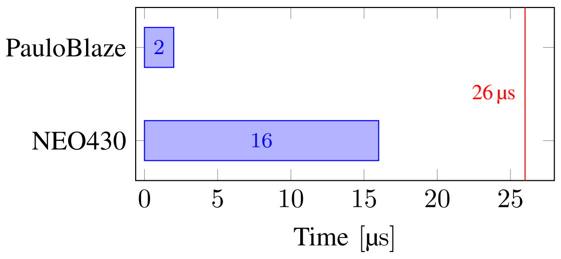

Fig. 8: Active time of the Softcores in the RIFS period compared to the 26 µs limit.

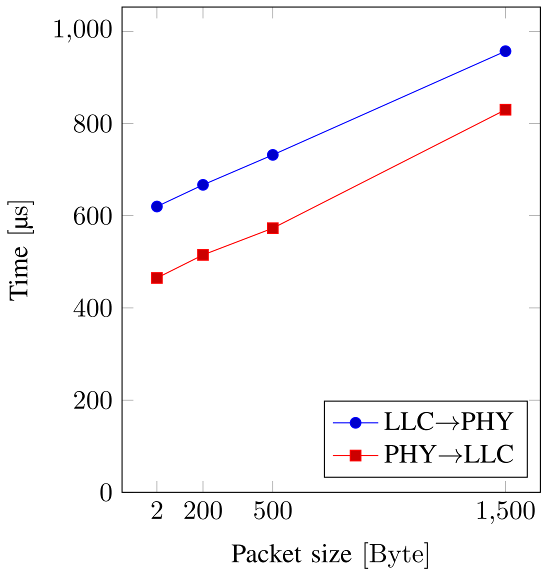

Fig. 9: Plot of the measured latency in Table III

TABLE III: Measured latency in the MAC layer

Packet size (Byte)	LLC→PHY (µs)	PHY→LLC (µs)
2	620	465
200	667	515
500	732	573
1500	957	830
Lin. Regression ($R^2 > 0,99$)	$0,2243x + 620$	$0,2437x + 461$

until the receiving station has to decide whether to send either a positive or negative acknowledgement. When the CPU requires less than 26 µs for this decision there is more room for possible adaptions and changes in the future. As shown in Fig. 8 this is the case for the PauloBlaze implementation. The NEO430 variant requires 16 µs, which is closer to the maximum. As the NEO430 is programmed in C, and the generated assembler code has shown to be suboptimal from time to time, further adaptions to the source code always require a close analysis of the resulting timing.

In a last step we evaluated the end-to-end throughput of the whole HomePlug system. For this experiment, two identical MAC layers were used. For the interconnection a simple PHY simulator was build that simulates an optimal, error-free channel. To measure the throughput, the time to send eight packages with 1.5 kB payload each was recorded. The transmission of the 12 kB needs 11.89 ms, which correspondents to a data rate of 8.07 Mbit/s. This value is comparable with the theoretical maximum of 8.4 Mbit/s and very close to simulations and experiments from other researchers, which measured a data rate of 8.08 Mbit/s [16].

The transmission rate on the channel is fixed by the HomePlug standard. Therefore, only the internal delays of the MAC layer limit the data throughput. To evaluate this latency, we measured two different delays: The time from packet availability on the LLC interface until the send request on the PHY interface (LLC→PHY) for the sender side and the delay from the first received byte on the PHY interface until the notification of the LLC (PHY→LLC) on the receiver side. Fig. 9 and Table III show these times for different package sizes. A linear trend is clearly visible. The slope of both graphs is about 0.23 µs/Byte, which is exactly the transfer rate of an internal copy loop in our software. This loop is used to copy packages from the LLC memory to the LFB (for the sender) or from the ReasMem to the LLC memory (for the receiver). The axis offset of 620 µs (LLC→PHY) and 461 µs (PHY→LLC) can be explained by the fixed amount of processing required for each package.

To reduce the linear part of the latency, the data transfer between the memories could be speed up using a dedicated DMA unit. Such a unit could possibly copy one byte per clock cycle, which results in a transfer rate of 20 ns/Byte for a 50 MHz clock. This would reduce the LLC→PHY delay for 1500 byte packages from 957 µs to 650 µs (-32.0 %). To reduce the static processing time, software optimisation might be possible. Another option is to run the CPU at a higher clock rate. Running at 100 MHz would cut the required time for processing the package in half, reducing the delay from 957 µs to 478.5 µs (-50 %).

VI. Conclusion

In this work we implemented a HomePlug 1.0.1 MAC layer for a Kintex 7 FPGA. We evaluated, if incorporating softcores in the design could lead to a more resource efficient solution. To take care of the special requirements of the MAC layer, two softcores were used in the design. One handles the complex communication with the LLC, the other one implements the real-time protocol of the PHY. Our analysis showed, that the implementation with softcores can be implemented with 78 % less FPGA resources compared to a pure FSM based implementation.

Measurements yield an end-to-end throughput of 8.07 Mbit/s, which is close to the theoretical maximum of 8.4 Mbit/s. Evaluation of the latency inside the MAC layer indicate that the current bottleneck is the data transfer between the LLC and the Local Frame Buffer. Using a DMA could reduce the required time by 32 %. Alternatively, the design could be run at an increased clock rate of 100 MHz, which would cut the latency by half.

To summarise, using softcores can make certain FPGA designs more resource-efficient while also increasing maintainability and flexibility.

References

[1] X. Li, R. Lu, X. Liang, X. Shen, J. Chen, and X. Lin, “Smart community: an internet of things application,” IEEE Communications Magazine, vol. 49, no. 11, pp. 68–75.

[2] Bluetooth Specification Version 4.2, Bluetooth SIG, 2014

[3] ZigBee Specification ZigBee Alliance, 2012

[4] IEEE Standard 802.11, Institute of Electrical and Electronics Engineers, 1997

[5] HomePlug 1.0 Specification, HomePlug Alliance Standard, Version 1.0.1, 2001

[6] HomePlug AV2 Technology, HomePlug Alliance Standard, 2010

[7] HomePlug Green PHY, HomePlug Alliance Standard, Version 1.1.1, 2013

[8] H. A. Latchman, S. Katar, L. Yonge, and S. Gavette, Homeplug AV and IEEE 1901: A Handbook for PLC Designers and Users, 1st ed. Wiley-IEEE Press.

[9] M. K. Lee et al.,“HomePlug 1.0 Powerline Communications LANs – Protocol Description and Performance Results,” International Journal of Commununication Systems, vol. 16, no. 5, pp. 447–473

[10] Dipl.-Ing. S. Nolting, “The NEO430 Processor,” 2017

[11] O. Girad, “openMSP430,” Rev. 1.16, 2016

[12] P. R. Genler, “PauloBlaze,” Technische Universitt Dresden, Deparment of Computer Science, Institute for Computer Engineering, Chair for VLSI Design, Diagnostic and Architecture

[13] https://opencores.org/project,avr_core

[14] C. Uhlemann, “Konzeptionierung, Implementierung und Verifikation eines MAC-Layers fr Paket-basierte Powerline Kommunikation,” 2017

[15] N. Rother, “Konzeptionierung und Implementierung einer hybriden MAC-Layer-Architektur fr Paket-basierte Powerline Kommunikation auf einem FPGA,” 2017

[16] M. K. Lee et al., “HomePlug 1.0 Powerline Communications LANs Protocol Description and Performance Results,” International Journal of Commununication Systems, vol. 16, no. 5, pp. 447–473

[17] ITU-T X.200 (07/1994), International Telecommunication Union