Mikrocomputertechnik 2		
Laborversuch LAN		
Name:	**Vorname:**	**Studiengang:**
	Datum:	**Testat:**

Ziel:

Nach Durchführung dieses Laborversuchs sollten Sie:

- verstehen, wie ein typischer Ethernetcontroller eines Mikrocomputers funktioniert
- verstehen, wie man eine typische Middleware benutzt, um Messwerte über die Ethernetschnittstelle unter Nutzung der TCP/IP-Protokollfamilie an einen anderen Computer zu übertragen oder von einem anderen Computer zu empfangen
- das Verhalten des Ethernet-Netzwerks mit Hilfe einer PC-Analysesoftware untersuchen und analysieren können

Einleitung

Die Fa. Keil, Hersteller der für den Labormikrocomputer genutzten Entwicklungsumgebung **µVision,** stellt das kompakte **Embedded-Betriebssystem RL-RTX** zur Verfügung, das für **Multitasking** und **Echtzeitanwendungen** geeignet ist.

Zu diesem Betriebsystem gibt es mehrere **Middlewaresoftwarepakete**, u.a. auch **MDK Middleware - Component ::Network**, in dem die TCP/IP Protokollfamilie implementiert ist.

Mit Hilfe des RTX-Betriebssystems und der MDK Middleware - Component ::Network soll der Labormikrocomputer in diesem Laborversuch im Multitasking zwei Aufgaben quasiparallel ausführen:

- Eine Messwerterfassung mit Übertragung der Messwerte über Ethernet
- Eine Messwertanzeige für über Ethernet empfangene Messwerte

Das Projekt **LAN** enthält die dafür benötigten Programm-Module.

Versuchsvorbereitung

1. Schritt: (nur notwendig, falls nicht schon beim früheren MCOM2-Laborversuch geschehen)

Legen Sie auf Ihrem persönlichen Laufwerk ein Verzeichnis **`MCOM`** an.
Kopieren Sie den vorbereiteten Laborprojektordner **`P:\normann\MCOM2\Labs`** vom Semesterlaufwerk in dieses Verzeichnis.

Das Unterverzeichnis **`LPC17xx`** enthält Dateien, die von allen MCOM-Laborprojekten mit dem M**ikrocomputer MCB1760** benutzt werden:

- Das Verzeichnis **`Drivers`** mit Treiberprogrammen

- Informationen zum MCB1760 sowie zum Mikrocontroller LPC1768 in pdf-Dateien

Im Unterverzeichnis **`\Labs\MCOM1\LAN`** befinden sich alle Dateien des eigentlichen Laborversuchs, das **µVision5-Projekt LAN.**

2. Schritt: (nur, falls 1. Schritt übersprungen wurde)

Kopieren Sie das Verzeichnis **`P:\normann\MCOM1\Labs\MCOM2\LAN`** in Ihr Verzeichnis MCOM2. Im Verzeichnis **LAN** befinden sich alle Dateien des eigentlichen Laborversuchs, das **µVision5-Projekt LAN.**

Aufgabe 1

Starten Sie µVision4 und öffnen Sie das **Projekt LAN** (Menü Project/OpenProject, Datei **LAN.uvproj**).

Das Projekt enthält 7 logische Gruppen von Programmdateien:

- **ApplicationSW** enthält die Anwendungsprogramme**.**
- **CTLabDrivers** enthält **`GPIOGroup.c`**, das Treiberprogramm zur Steuerung von Gruppen von Bitausgängen.
- **Board Support** enthält den AD-Wandler Treiber **`ADC.c`** und Treiber für das Grafikdisplay **`GLCD_MCB1700.c`**.
- **CMSIS** enthält das Konfigurationsprogramm **`RTX_Conf_CM.c`** und eine Bibliothek **`RTX_CM3.lib`** zum standardisierten Zugriff auf den RTOS Kernel.
- **CMSIS Driver** enthält die Treiber für die Ethernetkommunikation **`EMAC_LPC17xx.c`**, für den Ethernet Hardwaretreiber **`PHY_DP83848C.c`** und den SPI Treiber **`SSP_LPC17xx.c`**.
- **Device** enthält die Programme zum Konfigurieren des Microcontrollers LPC1768 beim Start: **`startup_LPC17xx.s`** und **`system_LPC17xx.c und`** Programme zur direkten Steuerung von I/O-Devices: **`GPIO_LPC17xx.c`**, **`PIN_LPC17xx.c`** und den DMA Treiber **`GPDMA_LPC17xx.c.`**
- **Network** enthält die Programme der **MDK Middleware - Component ::Network** zum Konfigurieren des Netzwerktreibers: **`Net_Config.c, Net_Config_ETH_0.h, Net_Config_TCP.h, Net_Config_UDP.h`** und die Bibliothek **`Net_CM3_L.lib.`**

In den Kommentarzeilen finden Sie einige Schlüsselwörter („**Tags**“), mit deren Hilfe das Freewaretool **Doxygen** (http://www.doxygen.org) eine Softwaredokumentation erstellt. Die Dokumentation kann über das µVision-Menü ***Tools/Doxygen*** erzeugt werden und wird im Unterverzeichnis **`.\Doc\html\`** des Projekts gespeichert. Sie ist mit einen Webbrowser lesbar (Doppelklick auf die Datei **`index.html`**).

Wählen Sie als Zielsystem „**LPC1768Flash**“ und lassen Sie das **Projekt übersetzen**. Kontrollieren Sie, ob keine Fehler gemeldet werden. Nutzen Sie die Doxygen-Dokumentation, um sich die **Aufgabe jedes Programm-Moduls klarzumachen**. Nutzen Sie ebenfalls die Klassen- und Sequenzdiagramme in den Anhängen.

Kontrollieren Sie, ob die **Sprachübersetzer** Fehler oder Warnungen melden.
Falls Fehler gemeldet werden, finden Sie die Ursache und korrigieren Sie das Programm.
Falls Warnungen gemeldet werden, finden Sie die Ursache und entscheiden Sie, ob eine Korrektur notwendig ist.

Der Linker generiert eine Textdatei **`LAN.map`**, aus der Sie entnehmen können, wie viel Speicherplatz das Gesamtprojekt LAN für Programm- und Datenspeicher benötigt.

Ermitteln Sie jeweils die Top3-Speicherfresser für Programmspeicher (Flash) und für Datenspeicher (RAM).

	Flash [Bytes]	**RAM [Bytes]**
Verfügbar:		
Benutzt:		
Rel. Auslastung [%]:		
Top 1:		
Top 2:		
Top 3:		

Der Linker generiert eine Datei **`LAN.htm`**, die Sie mit einem Webbrowser lesen können. (Das geht leider nicht innerhalb von µVision).
Diese Datei zeigt das Ergebnis einer statischen Analyse der **Schachtelungstiefe der Funktionsaufrufe und der maximal benutzten Stacktiefe („Static Call Graph“)**.

Nach erfolgreichem Build führt die integrierte Entwicklungsumgebung eine statische Analyse der **Schachtelungstiefe der Funktionsaufrufe** und der **maximal benutzten Stacktiefe** durch und speichert das Ergebnis **(„Static Call Graph“)** in einer Datei **`RTX.htm`**, die Sie mit einem Webbrowser lesen können.
(Das geht leider nicht innerhalb von µVision).

Bei Nutzung des RTX-Betriebssystems ist dabei zu beachten, dass verschiedene Stackspeicher eingerichtet werden:

- Jeder Einzelprozess bekommt einen eigenen, privaten ***Prozess-Stack***, dessen Größe in **`RTX_Conf_CM.c`** konfiguriert wird.

- Das Betriebssystem RTX und alle Interrupts nutzen den ***Mainstack***, der in **`startup_LPC17xx.s`** konfiguriert wird.

  Laut Datenblatt benötigt das RTX einen Stackspeicher von 128 Bytes.

  Bei der Aktivierung einer Interrupt-Service-Prozedur benutzt der Prozessor Cortex-M3 den Mainstackspeicher auch, um den Prozessorzustand zu speichern.
  Pro aktiviertem Interrupt werden dafür 32 Bytes verwendet.
  Dieser Stackbedarf addiert sich zum per statischer Analyse bestimmten Bedarf, da eine Unterbrechung zu jedem beliebigen Zeitpunkt auftreten kann.

Überprüfen Sie, ob im Interruptcontroller NVIC mehrere Interruptquellen unterschiedlicher Priorität konfiguriert sind, so dass geschachtelte Mehrfachunterbrechungen passieren können, da sich dann der notwendige Stackspeicher vervielfacht.

Mainstack	
Im NVIC konfigurierte Interruptquellen mit Prioritätsangabe:	
Maximale Schachteltiefe von Interrupts:	
Mainstackbedarf für Interrupts [Bytes]:	
Mainstackbedarf für RTX [Bytes]:	
Mainstackbedarf insgesamt [Bytes]:	
Konfigurierte Mainstackgröße [Bytes]:	

Prozessstacks

Prozesse	Benötigt [Bytes]	Konfiguriert [Bytes]

Kontrollieren Sie, ob die in **`startup_LPC17xx.s`** und in **`RTX_Conf_CM.c`** konfigurierten **Stackspeichergrößen** für das Gesamtprojekt ausreichen.

Aufgabe 2

Damit später die Ethernetkommunikation mit anderen Labormikrocomputern möglich ist, müssen Sie einige individuelle Konfigurationen an Ihrem Labormikrocomputer vornehmen.

Ändern Sie in **`Net_Config_ETH_0.h`** im Configuration Wizzard unter Ethernet Network Interface 0 das letzte Octet der **eigenen MAC-Address** auf (0x50 + Ihre Gruppennummer).
Ändern Sie auf die selbe Weise auch Ihre **eigene IP-Address** und übersetzen Sie das Projekt erneut.

Beispiel - Gruppe **7**:

MAC-Address:	`0x1E,0x30,0x6C,0xA2,0x45,0x5`**`7`**
IP-Address:	`192,168,0,10`**`7`**

Das Programm ist für eine Messwertübertragung mit **UDP-Protokoll** und **IP-Local Loopback (IP-Adresse 127.0.0.1)** als Empfänger vorkonfiguriert.
Auf diese Weise funktioniert das Programm, ohne dass der Mikrocomputer am Ethernet angeschlossen ist. Bei IP-Local-Loopback arbeitet der Sender Layer-3 direkt mit dem Empfänger Layer-3, so dass die Layer-2/Layer-1 Ethernethardware gar nicht benutzt wird.

Schließen Sie den Logikanalysator Digiview an die von Thread1 und Thread2 geschalteten LEDs an.
Starten Sie den Debugger und analysieren Sie mit Hilfe der LEDs und des Debuggers den zeitlichen Programmablauf.

Aufgabe 3

Schliessen Sie den Labormikrocomputer an das Labor-Ethernet an.

In diesem lokalen Ethernet sind alle Labormikrocomputer über einen **Hub** miteinander verbunden. Das Ethernet läuft also im klassischen, kollisionsbehafteten **Shared-Ethernet im Halb-Duplex-Betrieb**. Alle Ethernetpakete sind an allen Ethernetstationen sichtbar.
Am Dozenten-PC läuft die Analysesoftware **Wireshark**, mit der die Ethernetpakete aller Gruppen analysiert werden können.

Ändern Sie in **`ThreadComCfg.h`** in der Struktur **`tcpipConf`** die Empfänger-IP-Adresse in **IP-Netzwerk-Broadcast**, in diesem Fall also **192.168.0.255** und übersetzen Sie das Programm neu.

Wiederholen Sie die Messungen von Aufgabe 2 und analysieren Sie in Wireshark die von Ihnen gesendeten Ethernetframes.

Welche MAC-Adressen und welche IP-Protokolle werden benutzt?
Identifizieren Sie im Ethernet-Payload die Protokoll-Control Informationen für IP und UDP sowie den Inhalt des UDP-Payloads.
Vergleichen Sie den UDP-Payload mit den im Debugger ermittelten Messwerten.

Aufgabe 4

Bilden Sie mit Ihrer Nachbargruppe ein Team aus zwei Mikrocomputern, die sich gegenseitig Messwerte senden.

Ändern Sie dazu die von Ihnen benutzte Empfänger-IP-Adresse in die IP-Adresse Ihrer Nachbargruppe ab.
Schließen Sie jeweils einen Kanal des Logikanalysators Digiview an einen der beiden Mikrocomputer an, so dass Sie die Ereignisse beider Mikrocomputer zeitgleich zusammen messen können.

Wiederholen Sie die Messungen von Aufgabe 3 und analysieren Sie in Wireshark die von Ihnen gesendeten Ethernetframes.

Aufgabe 5

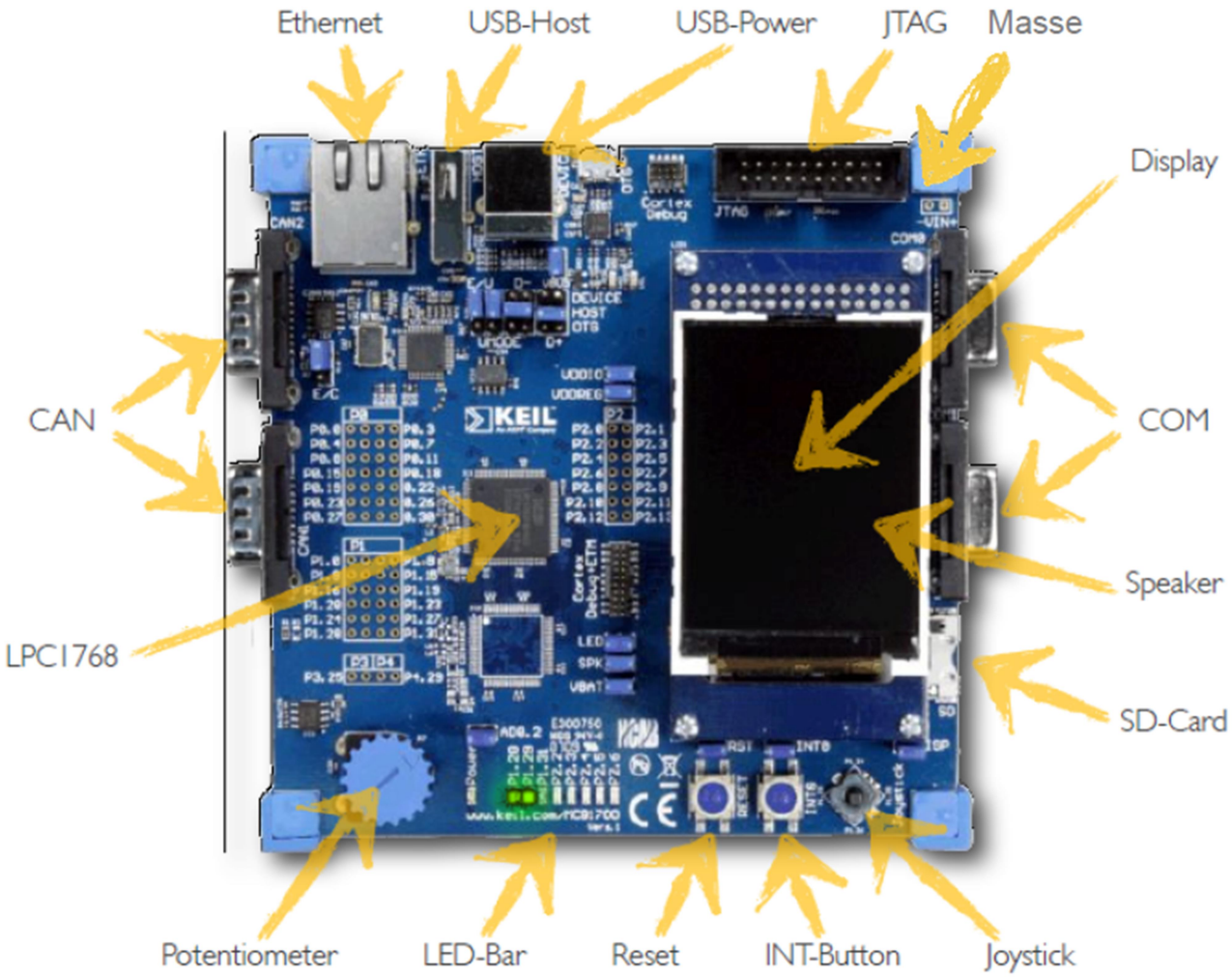
Alle bisherigen Messungen wurden mit dem UDP-Protokoll übertragen.

Ändern Sie in **`ThreadComCfg.h`** in der Struktur **`tcpipConf`** das benutzte Protokoll von **UDP** auf **TCP** und übersetzen Sie das Programm neu.

Wiederholen Sie die Messungen von Aufgabe 4 und analysieren Sie in Wireshark die von Ihnen gesendeten Ethernetframes.

Welche wesentlichen Änderungen fallen Ihnen auf?

Anhang 1: Keil MCB1760-ED Evaluation Board



Features

- 100MHz LPC1768 ARM Cortex™-M3 processor-based MCU in 100-pin LQFP
- On-Chip Memory: 512KB Flash & 64KB RAM
- Color QVGA TFT LCD
- 10/100 Ethernet Port
- USB 2.0 Full Speed - USB, USB-OTG, & USB Host
- 2 CAN Interfaces
- 2 Serial Ports
- SD/MMC Card Interface
- 5-position Joystick and push-button
- Analog Voltage Control for ADC Input
- Amplifier and Speaker
- 70 GPIO pins
- Debug Interface Connectors
  - 20 pin JTAG (0.1 inch connector)
  - 10 pin Cortex debug (0.05 inch connector)
  - 20 pin Cortex debug + ETM Trace (0.05 inch connector)

Anhang 2: Ethernetcontroller des LPC1768

Auszug aus **LPC1769/68/67/66/65/64/63 Product Data Sheet**:

8.11 Ethernet

Remark: The Ethernet controller is available on parts LPC1769/68/67/66/64. The Ethernet block supports bus clock rates of up to 100 MHz (LPC1768/67/66/64) or 120 MHz (LPC1769). See Table 2.

The **Ethernet block contains a full featured 10 Mbit/s or 100 Mbit/s Ethernet MAC** designed to provide optimized performance through the use of DMA hardware acceleration. Features include a generous suite of control registers, half or full duplex operation, flow control, control frames, hardware acceleration for transmit retry, receive packet filtering and wake-up on LAN activity. Automatic frame transmission and reception with scatter-gather DMA off-loads many operations from the CPU.

The **Ethernet block** and the **CPU share the ARM Cortex-M3 D-code and system bus** through the AHB-multilayer matrix to access the various on-chip SRAM blocks for Ethernet data, control, and status information.

The Ethernet block interfaces between an **off-chip Ethernet PHY** using the **Reduced MII (RMII) protoco**l and the on-chip **Media Independent Interface Management (MIIM)** serial bus.

8.11.1 Features

• Ethernet standards support:
– Supports 10 Mbit/s or 100 Mbit/s PHY devices including 10 Base-T, 100 Base-TX, 100 Base-FX, and 100 Base-T4.
– Fully compliant with *IEEE standard 802.3*.
– Fully compliant with 802.3x full duplex flow control and half duplex back pressure.
– Flexible transmit and receive frame options.
– Virtual Local Area Network (VLAN) frame support.

• Memory management:
– **Independent transmit** and **receive buffers memory mapped to shared SRAM**.
– DMA managers with scatter/gather DMA and arrays of frame descriptors.
– Memory traffic optimized by buffering and pre-fetching.

• Enhanced Ethernet features:
– Receive filtering.
– Multicast and broadcast frame support for both transmit and receive.
– Optional automatic Frame Check Sequence (FCS) insertion with Cyclic Redundancy Check (CRC) for transmit.
– Selectable automatic transmit frame padding.
– Over-length frame support for both transmit and receive allows any length frames.
– Promiscuous receive mode.
– Automatic collision back-off and frame retransmission.
– Includes power management by clock switching.
– Wake-on-LAN power management support allows system wake-up: using the receive filters or a magic frame detection filter.

• Physical interface:
– **Attachment of external PHY chip through standard RMII interface**.
– **PHY register access is available via the MIIM interface**.

Auszug aus **LPC17xx User manual**:

10.4 Architecture and operation

Figure 17 shows the internal architecture of the Ethernet block.

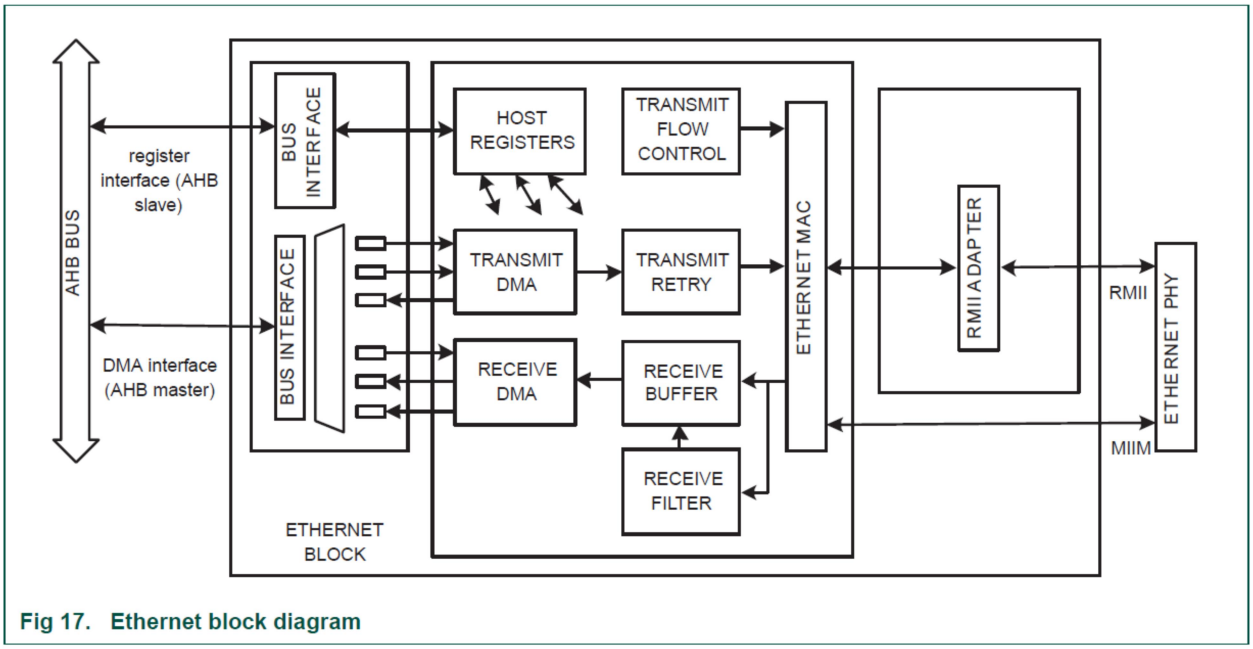


Fig 17. Ethernet block diagram

The block diagram for the Ethernet block consists of:

- The host registers module containing the registers in the software view and handling AHB accesses to the Ethernet block. The host registers connect to the transmit and receive data path as well as the MAC.

- The DMA to AHB interface. This provides an AHB master connection that allows the Ethernet block to access on-chip SRAM for reading of descriptors, writing of status, and reading and writing data buffers.

- The Ethernet MAC, which interfaces to the off-chip PHY via an RMII interface.

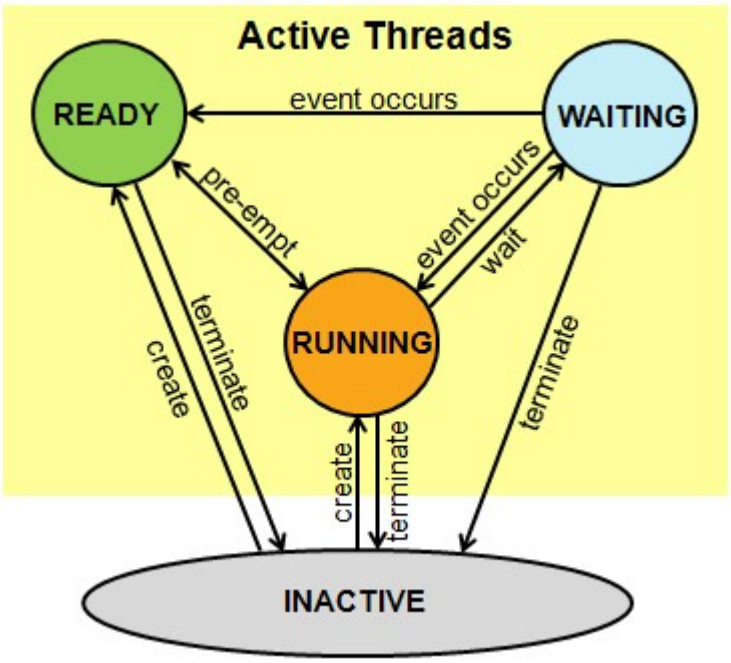
- The transmit data path, including:
  – The transmit DMA manager which reads descriptors and data from memory and writes status to memory.
  – The transmit retry module handling Ethernet retry and abort situations.
  – The transmit flow control module which can insert Ethernet pause frames.

- The receive data path, including:
  – The receive DMA manager which reads descriptors from memory and writes data and status to memory.
  – The Ethernet MAC which detects frame types by parsing part of the frame header.
  – The receive filter which can filter out certain Ethernet frames by applying different filtering schemes.
  – The receive buffer implementing a delay for receive frames to allow the filter to filter out certain frames before storing them to memory.

Anhang 3: Prozesszustände des RL-RTX

Threads can be in the following states:

- **RUNNING:** The thread that is currently running is in the **RUNNING** state. Only one thread at a time can be in this state.
- **READY:** Threads which are ready to run are in the READY state. Once the **RUNNING** thread has terminated or is **WAITING** the next READY thread with the highest priority becomes the RUNNING thread.
- **WAITING:** Threads that are waiting for an event to occur are in the **WAITING** state.
- **INACTIVE:** Threads that are not created or terminated are in the **INACTIVE** state. These threads typically consume no system resources.

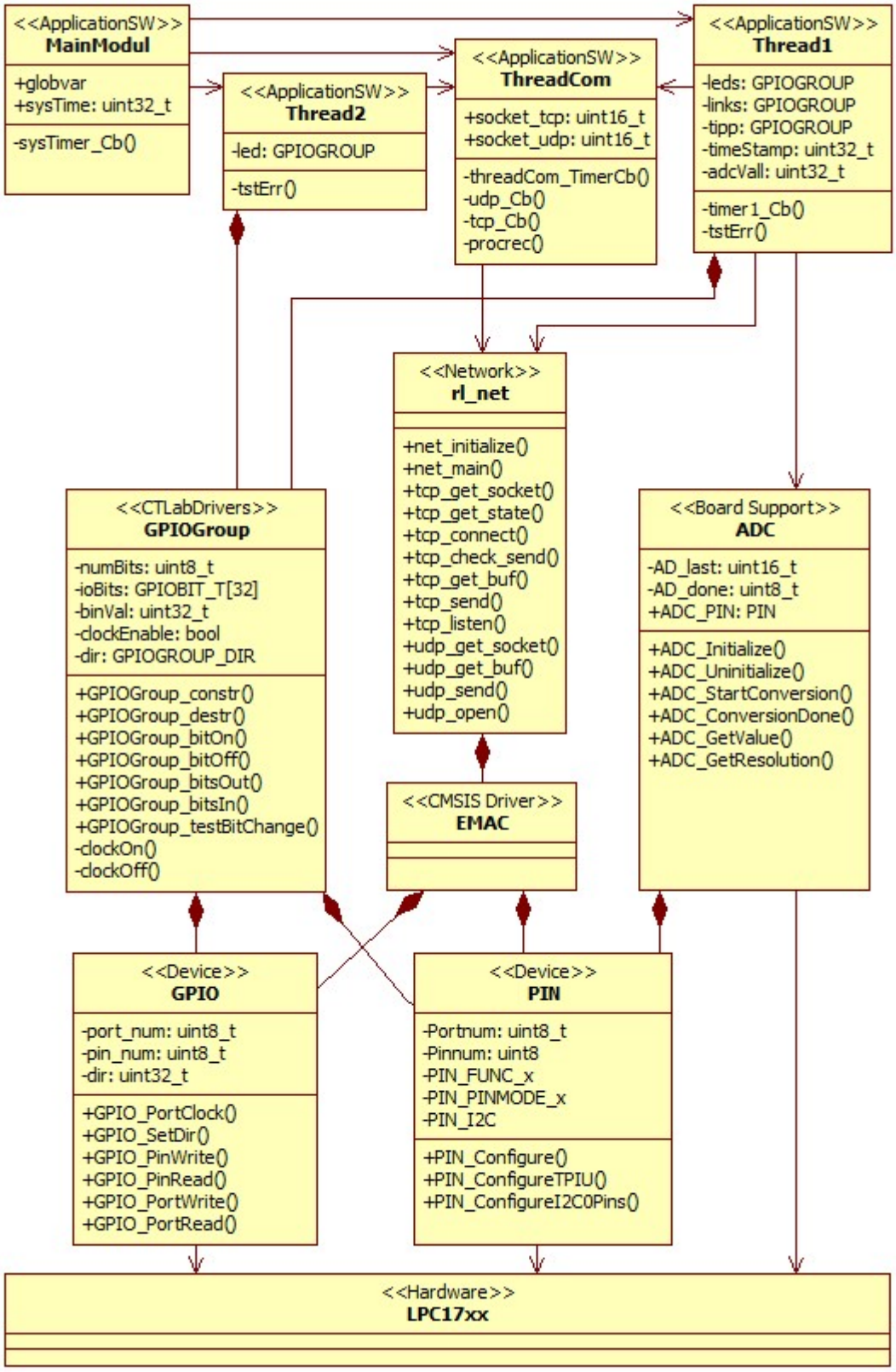


Thread State and State Transitions

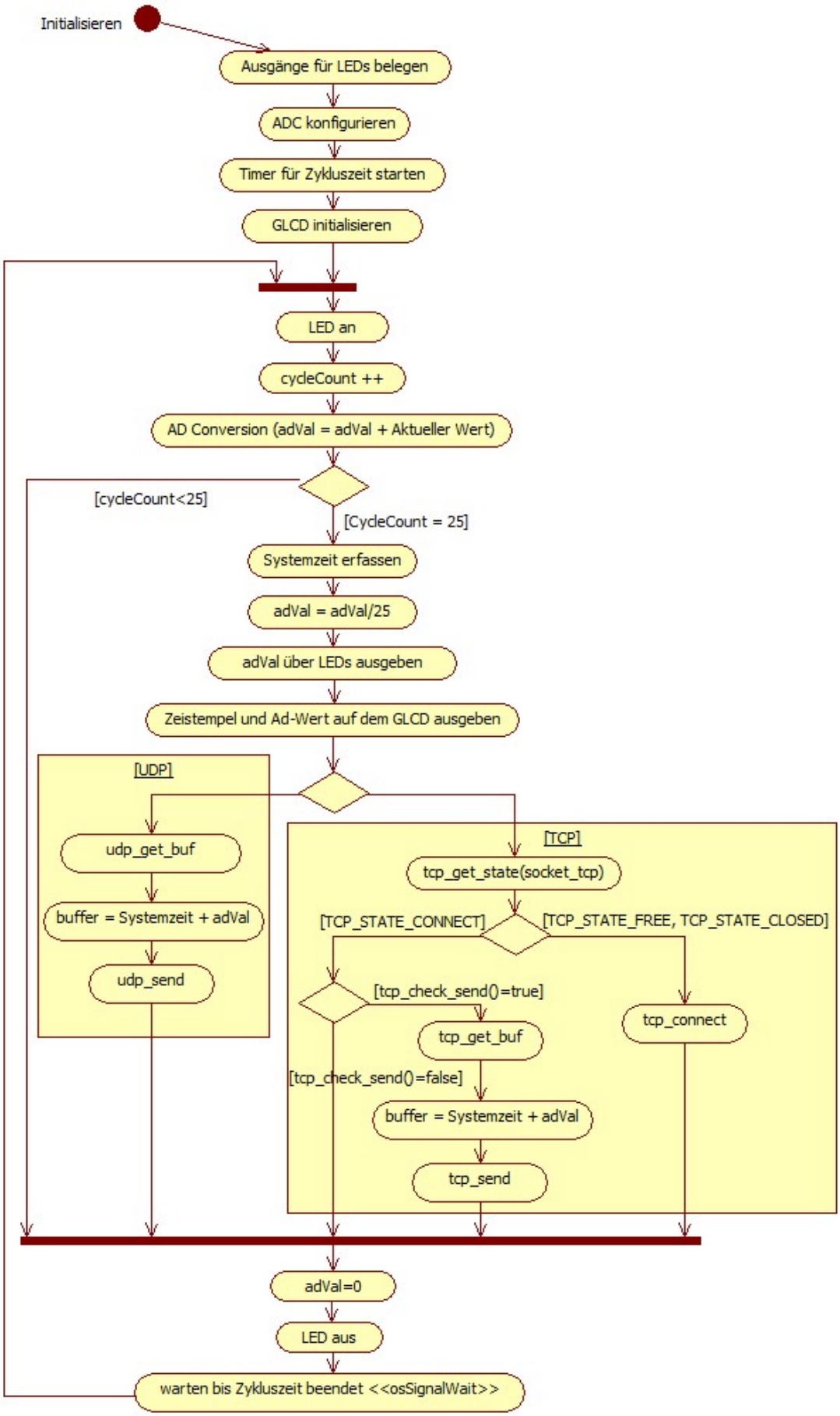
The CMSIS-RTOS assumes that threads are scheduled as shown in the figure **Thread State and State Transitions**. The thread states change as described below:

- A thread is created using the function osThreadCreate. This puts the thread into the **READY** or **RUNNING** state (depending on the thread priority).
- CMSIS-RTOS is pre-emptive. The active thread with the highest priority becomes the **RUNNING** thread provided it does not wait for any event. The initial priority of a thread is defined with the osThreadDef but may be changed during execution using the function osThreadSetPriority.
- The **RUNNING** thread transfers into the **WAITING** state when it is waiting for an event.
- Active threads can be terminated any time using the function osThreadTerminate. Threads can terminate also by just returning from the thread function. Threads that are terminated are in the **INACTIVE** state and typically do not consume any dynamic memory resources.

Anhang 4: UML Klassendiagramm Projekt LAN



Anhang 5: UML Aktivitätsdiagramm Process1



Anhang 6: UML Aktivitätsdiagramm Process2