

Duart Driver Nachschlagewerk  
v1.0

Erzeugt von Doxygen 1.3.9.1

Fri Feb 4 00:02:07 2005

## Inhaltsverzeichnis

<b>1</b>	<b>Duart Treiber Dokumentation für den MiniPAC MC332</b>	<b>1</b>
<b>2</b>	<b>Duart Driver Datenstruktur-Verzeichnis</b>	<b>2</b>
<b>3</b>	<b>Duart Driver Datei-Verzeichnis</b>	<b>2</b>
<b>4</b>	<b>Duart Driver Seitenindex</b>	<b>2</b>
<b>5</b>	<b>Duart Driver Datenstruktur-Dokumentation</b>	<b>3</b>
<b>6</b>	<b>Duart Driver Datei-Dokumentation</b>	<b>5</b>
<b>7</b>	<b>Duart Driver Zusätzliche Informationen</b>	<b>27</b>

## 1 Duart Treiber Dokumentation für den MiniPAC MC332

### Autor:

Dirk Hörner, Daniel Friedrich

### 1.1 Aufgabenstellung

Ziel des Projektes war es, einen Software-Treiber für den Philips SC28L92 Dual Asynchronous Receiver/Transmitter (DUART) auf Basis des MiniPAC 323 Microcontrollers zu entwickeln. Der Treiber soll die Initialisierung, den Datentransfer, die Konfiguration, die Diagnose und die Fehlerbehandlung vereinfachen.

### 1.2 Erreichte Ziele

An erster Stelle des Entwurfes des Treibers stand die Einfachheit der Konfiguration des Chips. Obwohl es relativ schwer war, die komplexen Einstellungsmöglichkeiten in einfachen Strukturen zu kapseln, ist dies zufriedenstellend gelungen. Zwar musste Aufgrund von Zeitmangel auf manche Optionen verzichtet werden, aber die grundlegenden und meist genutzten Parameter sind bequem einstellbar.

Weiterhin wurde für den Treiber eine eigene FIFO als Ringpuffer implementiert, um den Zugriff auf empfangene Daten zu vereinfachen und um eine größere Vorhaltezeit zu bieten (Größe des FIFOs im Quellcode anpassbar).

Zusätzlich wird dem Programmierer die Freiheit gegeben, sich selber um die Interruptbehandlung zu kümmern, oder den Treiber im Polling- bzw. Blocking-Mode zu benutzen.

## 1.3 Mögliche Zukunftspläne

Im Zuge weiterer Projekte könnte beim Treiber die Konfigurierbarkeit erhöht werden (Bits-per-Character, Watchdog, Parityfehler-Behandlung, ...), die Statusabfrage von Kanälen/des Chips ermöglicht werden und die Einstellbarkeit des Interrupt-Event-Handling verbessert werden (Interrupt-Maske, Interrupt-Status, ...).

## 2 Duart Driver Datenstruktur-Verzeichnis

### 2.1 Duart Driver Datenstrukturen

Hier folgt die Aufzählung aller Datenstrukturen mit einer Kurzbeschreibung:

<code>duart_channel_t</code> (Hält alle Konfigurationsdaten für einen Kanal vom Chip )	3
<code>duart_fifo_t</code> (Diese Struktur hält alle wichtigen Information für jeweils eine FIFO )	4

## 3 Duart Driver Datei-Verzeichnis

### 3.1 Duart Driver Auflistung der Dateien

Hier folgt die Aufzählung aller dokumentierten Dateien mit einer Kurzbeschreibung:

<code>duart_baudrates.h</code> (Alle Baudraten die vom Chip unterstützt werden )	5
<code>duart_driver.h</code> (Alle wesentlichen defines, Strukturen und Funktionen des Treibers )	7
<code>duart_fifo.h</code> (Implementierung einer Ringpuffer-FIFO für byteweises Speichern und Auslesen )	22
<code>duart_registers.h</code> (Die Adressen zu den Registern des Chipsatzes werden festgelegt )	26

## 4 Duart Driver Seitenindex

### 4.1 Duart Driver Zusätzliche Informationen

Hier folgt eine Liste mit zusammengehörigen Themengebieten:

Liste der zu erledigenden Dinge	27
---------------------------------	----

## 5 Duart Driver Datenstruktur-Dokumentation

### 5.1 duart\_channel\_t Strukturreferenz

Hält alle Konfigurationsdaten für einen Kanal vom Chip.

#### Datenfelder

- unsigned char **channel**  
*Der zu benutzende Kanal (DUART\_CHANNEL\_\*).*
- unsigned char **enable**  
*Die Rx/Tx Schalter (DUART\_(EN|DIS)ABLE\_\*).*
- unsigned char **baudrate**  
*Die zu benutzende Baudrate (DUART\_B\*).*
- unsigned char **parity**  
*Der zu benutzende Parity-Mode (DUART\_PARITY\_\*).*
- unsigned char **stopbits**  
*Die Anzahl der Stopbits (DUART\_STOPBITS\*).*
- unsigned char **channel\_mode**  
*Der Kanal-Modus (DUART\_CHANMODE\_\*).*
- **duart\_fifo\_t \* fifo**  
*Pointer zum FIFO des Kanals.*

#### 5.1.1 Ausführliche Beschreibung

Hält alle Konfigurationsdaten für einen Kanal vom Chip.

#### 5.1.2 Dokumentation der Datenelemente

##### 5.1.2.1 unsigned char duart\_channel\_t::parity

Der zu benutzende Parity-Mode (DUART\_PARITY\_\*).

Die parity-Variable ist eine bitweise Verknüpfung (oder) aus:

DUART\_PARITY\_EVEN, DUART\_PARITY\_ODD (Art der Parity)

und

DUART\_PARITY\_MODE\_\*

Wird `PARITY_MODE_NO` gewählt, haben `DUART_PARITY_EVEN` bzw. `DUART_PARITY_ODD` keine Wirkung.

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- `duart_driver.h`

## 5.2 `duart_fifo_t` Strukturreferenz

Diese Struktur hält alle wichtigen Information für jeweils eine FIFO.

### Datenfelder

- unsigned char **overflow**  
*Anzeige ob overflow stattgefunden hat seit dem letzten zurücksetzen der FIFO.*
- unsigned int **size**  
*Größe des FIFO-Puffers in Byte.*
- char \* **buf**  
*Zeiger auf das erste Byte im FIFO-Puffer.*
- char \* **next\_read**  
*Zeiger auf das nächste zu lesende Byte im Puffer.*
- char \* **next\_elem**  
*Zeiger auf das nächste zu schreibende Byte im Puffer.*
- unsigned int **numbytes**  
*Anzahl der noch nicht gelesenen Bytes im Puffer.*

### 5.2.1 Ausführliche Beschreibung

Diese Struktur hält alle wichtigen Information für jeweils eine FIFO.

### 5.2.2 Dokumentation der Datenelemente

#### 5.2.2.1 unsigned char `duart_fifo_t::overflow`

Anzeige ob overflow stattgefunden hat seit dem letzten zurücksetzen der FIFO.

Ein overflow wird ausgelöst wenn einmal um den Ringpuffer herum geschrieben wurde (ausgehend von der Stelle des letzten Lesezugriffs) und erneut ein Byte in die FIFO geschrieben werden soll (noch nicht gelesene Daten werden überschrieben).

Die overflow-Variable wird nach einem `duart_fifo_reset()`(S. 23) zurück gesetzt.

**Achtung:**

Der Programmierer muss die overflow-Variable selbstständig zurücksetzen, es sei denn `duart_fifo_reset()`(S. 23) wird aufgerufen.

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- `duart_fifo.h`

## 6 Duart Driver Datei-Dokumentation

### 6.1 `duart_baudrates.h`-Dateireferenz

Alle Baudraten die vom Chip unterstützt werden.

#### Makrodefinitionen

- `#define DUART_B50 0`  
*50 Baud*
- `#define DUART_B75 1`  
*75 Baud*
- `#define DUART_B110 2`  
*110 Baud*
- `#define DUART_B134_5 3`  
*134,5 Baud*
- `#define DUART_B150 4`  
*150 Baud*
- `#define DUART_B200 5`  
*200 Baud*
- `#define DUART_B300 6`  
*300 Baud*
- `#define DUART_B600 7`  
*600 Baud*
- `#define DUART_B1050 8`  
*1050 Baud*

- #define **DUART\_B1200** 9  
*1200 Baud*
- #define **DUART\_B1800** 10  
*1800 Baud*
- #define **DUART\_B2000** 11  
*2000 Baud*
- #define **DUART\_B2400** 12  
*2400 Baud*
- #define **DUART\_B4800** 13  
*4800 Baud*
- #define **DUART\_B7200** 14  
*7200 Baud*
- #define **DUART\_B9600** 15  
*9600 Baud*
- #define **DUART\_B19200** 16  
*19200 Baud*
- #define **DUART\_B38400** 17  
*38400 Baud*
- #define **DUART\_B57600** 18  
*57600 Baud*
- #define **DUART\_B115200** 19  
*115200 Baud*
- #define **DUART\_B230400** 20  
*230400 Baud*

## Variablen

- unsigned char **duart\_br** [21][3]

### 6.1.1 Ausführliche Beschreibung

Alle Baudraten die vom Chip unterstützt werden.

## 6.2 `duart_driver.h`-Dateireferenz

Alle wesentlichen defines, Strukturen und Funktionen des Treibers.

### Datenstrukturen

- struct `duart_channel_t`  
*Hält alle Konfigurationsdaten für einen Kanal vom Chip.*

### Makrodefinitionen

- #define `DUART_GET_REG(addr)` `((unsigned char *) (duart_base_addr + addr))`  
*Gibt den Inhalt des Registers an der Adresse `addr` zurück.*
- #define `DUART_SET_REG(addr, data)` `((unsigned char *) (duart_base_addr + addr)) = data`  
*Setzt den Inhalt eines Registers an der Adresse `addr` auf den Wert `data`.*
- #define `DUART_TX_READY(chan)` `(DUART_GET_REG(REG_ADDR_STATUS + chan → channel) & 0x04)`  
*Überprüft ob Daten im Chip-Sende-FIFO liegen.*
- #define `DUART_RX_READY(chan)` `(DUART_GET_REG(REG_ADDR_STATUS + chan → channel) & 0x01)`  
*Überprüft ob Daten im Chip-Empfangs-FIFO liegen.*
- #define `DUART_CHANNEL_A` `0x00`  
*Kanal A.*
- #define `DUART_CHANNEL_B` `0x10`  
*Kanal B.*
- #define `DUART_ENABLE_TX` `0x4`  
*Aktiviert den Transmitter.*
- #define `DUART_DISABLE_TX` `0x8`  
*Deaktiviert den Transmitter.*
- #define `DUART_ENABLE_RX` `0x1`  
*Aktiviert den Empfänger.*
- #define `DUART_DISABLE_RX` `0x2`  
*Deaktiviert den Empfänger.*

- #define **DUART\_PARITY\_EVEN** 0x0  
*Gerade Parity.*
- #define **DUART\_PARITY\_ODD** 0x4  
*Ungerade Parity.*
- #define **DUART\_PARITY\_MODE\_WITH** 0x0  
*Aktiviert Parity.*
- #define **DUART\_PARITY\_MODE\_FORCE** 0x8  
*Forciert Parity.*
- #define **DUART\_PARITY\_MODE\_NO** 0x10  
*Keine Parity benutzen.*
- #define **DUART\_PARITY\_MODE\_MULTIDROP** 0x18  
*Spezieller Multidrop-Modus für Kanal A.*
- #define **DUART\_STOPBITS\_0\_563** 0x0  
*Stopbitlänge: 0,563.*
- #define **DUART\_STOPBITS\_0\_625** 0x1  
*Stopbitlänge: 0,625.*
- #define **DUART\_STOPBITS\_0\_688** 0x2  
*Stopbitlänge: 0,688.*
- #define **DUART\_STOPBITS\_0\_750** 0x3  
*Stopbitlänge: 0,750.*
- #define **DUART\_STOPBITS\_0\_813** 0x4  
*Stopbitlänge: 0,813.*
- #define **DUART\_STOPBITS\_0\_875** 0x5  
*Stopbitlänge: 0,875.*
- #define **DUART\_STOPBITS\_0\_938** 0x6  
*Stopbitlänge: 0,938.*
- #define **DUART\_STOPBITS\_1** 0x7  
*Stopbitlänge: 1.*
- #define **DUART\_STOPBITS\_1\_563** 0x8  
*Stopbitlänge: 1,563.*
- #define **DUART\_STOPBITS\_1\_625** 0x9

*Stopbitlänge: 1,625.*

- `#define DUART_STOPBITS_1_688 0xA`  
*Stopbitlänge: 1,688.*
- `#define DUART_STOPBITS_1_750 0xB`  
*Stopbitlänge: 1,750.*
- `#define DUART_STOPBITS_1_813 0xC`  
*Stopbitlänge: 1,813.*
- `#define DUART_STOPBITS_1_875 0xD`  
*Stopbitlänge: 1,875.*
- `#define DUART_STOPBITS_1_938 0xE`  
*Stopbitlänge: 1,938.*
- `#define DUART_STOPBITS_2 0xF`  
*Stopbitlänge: 2.*
- `#define DUART_STOPBITS5_1_063 0x0`  
*Stopbitlänge für 5bit Datenlänge: 1,063.*
- `#define DUART_STOPBITS5_1_125 0x1`  
*Stopbitlänge für 5bit Datenlänge: 1,125.*
- `#define DUART_STOPBITS5_1_188 0x2`  
*Stopbitlänge für 5bit Datenlänge: 1,188.*
- `#define DUART_STOPBITS5_1_250 0x3`  
*Stopbitlänge für 5bit Datenlänge: 1,250.*
- `#define DUART_STOPBITS5_1_313 0x4`  
*Stopbitlänge für 5bit Datenlänge: 1,313.*
- `#define DUART_STOPBITS5_1_375 0x5`  
*Stopbitlänge für 5bit Datenlänge: 1,375.*
- `#define DUART_STOPBITS5_1_438 0x6`  
*Stopbitlänge für 5bit Datenlänge: 1,438.*
- `#define DUART_STOPBITS5_1_5 0x7`  
*Stopbitlänge für 5bit Datenlänge: 1,5.*
- `#define DUART_CHANMODE_NORMAL 0x00;`  
*Normaler Modus, der Transmitter und der Empfänger arbeiten getrennt voneinander.*

- #define **DUART\_CHANMODE\_AUTO\_ECHO** 0x40;  
*Autoecho Modus, der Chip sendet alle empfangenen Daten automatisch zurück.*
- #define **DUART\_CHANMODE\_LOCAL\_LOOP** 0x80;  
*local loop, transmitter output is internally connected to the receiver*
- #define **DUART\_CHANMODE\_REMOTE\_LOOP** 0xC0;  
*remote loop, received data is reclocked and retransmitted on the TxDA output*
- #define **FIFOSIZE** 512  
*Die Treiber-FIFO-Größe (nur Empfang) in Byte.*
- #define **IntVec** 27
- #define **INTERRUPT\_DISABLE** asm(" oriw #0x700,%sr")
- #define **INTERRUPT\_ENABLE** asm(" movew #0x2000,%sr")

## Funktionen

- void **duart\_init** (unsigned int base\_addr)  
*Initialisiert den Duart-Chip und mapt alle seine Register in den Hauptspeicher angefangen bei der Basis-Speicheradresse base\_addr.*
- void **duart\_write** (duart\_channel\_t \*chan, char \*data, int size)  
*Schreibt size Bytes angefangen bei data in den Kanal chan.*
- char **duart\_getchar** (duart\_channel\_t \*chan)  
*Liest ein Byte aus dem Kanal chan aus (blockend).*
- void **duart\_channel\_setattr** (duart\_channel\_t \*chan)  
*Konfiguriert den angegebenen Kanal mit den übergebenen Daten.*
- void **duart\_channel\_getattr** (duart\_channel\_t \*chan)  
*Liest die Einstellungen vom Chip in die übergebene duart\_channel\_t(S.3) Struktur.*
- **duart\_channel\_t \* duart\_create\_channel** ()  
*Legt eine duart\_channel\_t(S.3) Struktur an und füllt sie mit Standardwerten.*
- void **duart\_destroy\_channel** (duart\_channel\_t \*chan)  
*Löscht die übergebene duart\_channel\_t(S.3) Struktur vom Heap.*
- **\_\_attribute\_\_((interrupt)) static void duart\_isr**(void)  
*Interrupt Service Routine für Interrupt-Event-Handling.*

- void **duart\_init\_isr** (char IntLvl, void(\*isr)(void))  
*Initialisiert das Interrupt-Event-Handling.*

### Variablen

- void(\* **IVT** [256])(void)

### 6.2.1 Ausführliche Beschreibung

Alle wesentlichen defines, Strukturen und Funktionen des Treibers.

### 6.2.2 Makro-Dokumentation

#### 6.2.2.1 #define DUART\_GET\_REG(addr) (\*((unsigned char \*) (duart\_base\_addr + addr)))

Gibt den Inhalt des Registers an der Adresse addr zurück.

Damit dieses Makro funktioniert, muss erst **duart\_init()**(S.18) aufgerufen werden.

#### 6.2.2.2 #define DUART\_SET\_REG(addr, data) (\*((unsigned char \*) (duart\_base\_addr + addr)) = data)

Setzt den Inhalt eines Registers an der Adresse addr auf den Wert data.

Damit dieses Makro funktioniert, muss erst **duart\_init()**(S.18) aufgerufen werden.

#### 6.2.2.3 #define DUART\_CHANNEL\_A 0x00

Kanal A.

Siehe auch:

**duart\_channel\_t::channel**(S.3)

#### 6.2.2.4 #define DUART\_CHANNEL\_B 0x10

Kanal B.

Siehe auch:

**duart\_channel\_t::channel**(S.3)

**6.2.2.5 #define DUART\_ENABLE\_TX 0x4**

Aktiviert den Transmitter.

Siehe auch:

`duart_channel_t::enable`(S. 3)

**6.2.2.6 #define DUART\_DISABLE\_TX 0x8**

Deaktiviert den Transmitter.

Siehe auch:

`duart_channel_t::enable`(S. 3)

**6.2.2.7 #define DUART\_ENABLE\_RX 0x1**

Aktiviert den Empfänger.

Siehe auch:

`duart_channel_t::enable`(S. 3)

**6.2.2.8 #define DUART\_DISABLE\_RX 0x2**

Deaktiviert den Empfänger.

Siehe auch:

`duart_channel_t::enable`(S. 3)

**6.2.2.9 #define DUART\_PARITY\_EVEN 0x0**

Gerade Parity.

Siehe auch:

`duart_channel_t::parity`(S. 3)

**6.2.2.10 #define DUART\_PARITY\_ODD 0x4**

Ungerade Parity.

Siehe auch:

`duart_channel_t::parity`(S. 3)

**6.2.2.11 #define DUART\_PARITY\_MODE\_WITH 0x0**

Aktiviert Parity.

**Siehe auch:**

`duart_channel_t::parity`(S. 3)

**6.2.2.12 #define DUART\_PARITY\_MODE\_FORCE 0x8**

Forciert Parity.

**Siehe auch:**

`duart_channel_t::parity`(S. 3)

**6.2.2.13 #define DUART\_PARITY\_MODE\_NO 0x10**

Keine Parity benutzen.

**Siehe auch:**

`duart_channel_t::parity`(S. 3)

**6.2.2.14 #define DUART\_PARITY\_MODE\_MULTIDROP 0x18**

Spezieller Multidrop-Modus für Kanal A.

**Siehe auch:**

`duart_channel_t::parity`(S. 3)

**6.2.2.15 #define DUART\_STOPBITS\_0\_563 0x0**

Stopbitlänge: 0,563.

**Siehe auch:**

`duart_channel_t::stopbits`(S. 3)

**6.2.2.16 #define DUART\_STOPBITS\_0\_625 0x1**

Stopbitlänge: 0,625.

**Siehe auch:**

`duart_channel_t::stopbits`(S. 3)

**6.2.2.17 #define DUART\_STOPBITS\_0\_688 0x2**

Stopbitlänge: 0,688.

Siehe auch:

`duart_channel_t::stopbits`(S. 3)

**6.2.2.18 #define DUART\_STOPBITS\_0\_750 0x3**

Stopbitlänge: 0,750.

Siehe auch:

`duart_channel_t::stopbits`(S. 3)

**6.2.2.19 #define DUART\_STOPBITS\_0\_813 0x4**

Stopbitlänge: 0,813.

Siehe auch:

`duart_channel_t::stopbits`(S. 3)

**6.2.2.20 #define DUART\_STOPBITS\_0\_875 0x5**

Stopbitlänge: 0,875.

Siehe auch:

`duart_channel_t::stopbits`(S. 3)

**6.2.2.21 #define DUART\_STOPBITS\_0\_938 0x6**

Stopbitlänge: 0,938.

Siehe auch:

`duart_channel_t::stopbits`(S. 3)

**6.2.2.22 #define DUART\_STOPBITS\_1 0x7**

Stopbitlänge: 1.

Siehe auch:

`duart_channel_t::stopbits`(S. 3)

**6.2.2.23 #define DUART\_STOPBITS\_1\_563 0x8**

Stopbitlänge: 1,563.

Siehe auch:

`duart_channel_t::stopbits`(S. 3)

**6.2.2.24 #define DUART\_STOPBITS\_1\_625 0x9**

Stopbitlänge: 1,625.

Siehe auch:

`duart_channel_t::stopbits`(S. 3)

**6.2.2.25 #define DUART\_STOPBITS\_1\_688 0xA**

Stopbitlänge: 1,688.

Siehe auch:

`duart_channel_t::stopbits`(S. 3)

**6.2.2.26 #define DUART\_STOPBITS\_1\_750 0xB**

Stopbitlänge: 1,750.

Siehe auch:

`duart_channel_t::stopbits`(S. 3)

**6.2.2.27 #define DUART\_STOPBITS\_1\_813 0xC**

Stopbitlänge: 1,813.

Siehe auch:

`duart_channel_t::stopbits`(S. 3)

**6.2.2.28 #define DUART\_STOPBITS\_1\_875 0xD**

Stopbitlänge: 1,875.

Siehe auch:

`duart_channel_t::stopbits`(S. 3)

**6.2.2.29 #define DUART\_STOPBITS\_1\_938 0xE**

Stopbitlänge: 1,938.

Siehe auch:

`duart_channel_t::stopbits`(S. 3)

**6.2.2.30 #define DUART\_STOPBITS\_2 0xF**

Stopbitlänge: 2.

Siehe auch:

`duart_channel_t::stopbits`(S. 3)

**6.2.2.31 #define DUART\_STOPBITS5\_1\_063 0x0**

Stopbitlänge für 5bit Datenlänge: 1,063.

Siehe auch:

`duart_channel_t::stopbits`(S. 3)

**6.2.2.32 #define DUART\_STOPBITS5\_1\_125 0x1**

Stopbitlänge für 5bit Datenlänge: 1,125.

Siehe auch:

`duart_channel_t::stopbits`(S. 3)

**6.2.2.33 #define DUART\_STOPBITS5\_1\_188 0x2**

Stopbitlänge für 5bit Datenlänge: 1,188.

Siehe auch:

`duart_channel_t::stopbits`(S. 3)

**6.2.2.34 #define DUART\_STOPBITS5\_1\_250 0x3**

Stopbitlänge für 5bit Datenlänge: 1,250.

Siehe auch:

`duart_channel_t::stopbits`(S. 3)

**6.2.2.35 #define DUART\_STOPBITS5\_1\_313 0x4**

Stopbitlänge für 5bit Datenlänge: 1,313.

Siehe auch:

`duart_channel_t::stopbits`(S. 3)

**6.2.2.36 #define DUART\_STOPBITS5\_1\_375 0x5**

Stopbitlänge für 5bit Datenlänge: 1,375.

Siehe auch:

`duart_channel_t::stopbits`(S. 3)

**6.2.2.37 #define DUART\_STOPBITS5\_1\_438 0x6**

Stopbitlänge für 5bit Datenlänge: 1,438.

Siehe auch:

`duart_channel_t::stopbits`(S. 3)

**6.2.2.38 #define DUART\_STOPBITS5\_1\_5 0x7**

Stopbitlänge für 5bit Datenlänge: 1,5.

Siehe auch:

`duart_channel_t::stopbits`(S. 3)

**6.2.2.39 #define DUART\_CHANMODE\_NORMAL 0x00;**

Normaler Modus, der Transmitter und der Empfänger arbeiten getrennt voneinander.

Siehe auch:

`duart_channel_t::channel_mode`(S. 3)

**6.2.2.40 #define DUART\_CHANMODE\_AUTO\_ECHO 0x40;**

Autoecho Modus, der Chip sendet alle empfangenen Daten automatisch zurück. TxRDY und TxEMT status bits sind inaktiv (0)

Siehe auch:

`duart_channel_t::channel_mode`(S. 3)

**6.2.2.41 #define DUART\_CHANMODE\_LOCAL\_LOOP 0x80;**

local loop, transmitter output is internally connected to the receiver

Siehe auch:

`duart_channel_t::channel_mode`(S. 3)

**6.2.2.42 #define DUART\_CHANMODE\_REMOTE\_LOOP 0xC0;**

remote loop, received data is reclocked and retransmitted on the TxDA output

Siehe auch:

`duart_channel_t::channel_mode`(S. 3)

**6.2.3 Dokumentation der Funktionen****6.2.3.1 void duart\_init (unsigned int base\_addr)**

Initialisiert den Duart-Chip und mapt alle seine Register in den Hauptspeicher angefangen bei der Basis-Speicheradresse `base_addr`.

Parameter:

*base\_addr* Die Basis-Speicheradresse zu welcher die Register gemapt werden sollen.

```

13 {
14     duart_base_addr = base_addr;
15
16     CSPAR1 = 0x03FF; /* pin assignment register, prepare for chipselect 7 */
17     CSBAR7 = duart_base_addr >> 8; /* base address */
18
19     /* MODE    async    0
20      * BYTE    both     11
21      * R/Wn    both     11
22      * STRB    DSn      1
23      * DSACK   1 Wait   0001
24      * SPACE   S/U SP   11
25      * IPL     3        011
26      * AVECn   on       1
27      */
28     CSOR7 = 0x7C77;
29
30     /* initialize the fifos */
31     fifos[0] = duart_fifo_create(FIFOSIZE);
32     assert(fifos[0]);
33     fifos[1] = duart_fifo_create(FIFOSIZE);
34     assert(fifos[1]);
35 }
```

**6.2.3.2 void duart\_write (duart\_channel\_t \* chan, char \* data, int size)**

Schreibt `size` Bytes angefangen bei `data` in den Kanal `chan`.

**Parameter:**

*chan* Kanal in den geschrieben werden soll

*data* Zeiger auf die Daten

*size* Anzahl der Bytes die geschrieben werden sollen

```

38 {
39     int i;
40
41     for (i=0; i < size; i++)
42     {
43         /* auf TxRDY warten */
44         while (!DUART_TX_READY(chan));
45
46         /* byte schreiben */
47         DUART_SET_REG(REG_ADDR_TXFIFO + chan->channel,
48             data[i]);
49     }
50 }

```

**6.2.3.3 char duart\_getchar (duart\_channel\_t \* chan)**

Liest ein Byte aus dem Kanal chan aus (blockend).

**Parameter:**

*chan* der Kanal aus dem gelesen werden soll

**Rückgabe:**

das gelesene Byte

```

53 {
54     while (!DUART_RX_READY(chan));
55
56     return DUART_GET_REG(REG_ADDR_RXFIFO + chan->channel);
57 }

```

**6.2.3.4 void duart\_channel\_setattr (duart\_channel\_t \* chan)**

Konfiguriert den angegebenen Kanal mit den übergebenen Daten.

**Parameter:**

*chan* Zeiger auf die **duart\_channel\_t**(S.3) Struktur

```

64 {
65     char mr0;
66
67     /* reset the channel */
68     DUART_SET_REG(REG_ADDR_COMMAND + chan->channel, 0x20); /* recv reset */
69     DUART_SET_REG(REG_ADDR_COMMAND + chan->channel, 0x30); /* trans reset */
70
71     DUART_SET_REG(REG_ADDR_COMMAND + chan->channel, 0xB0); /* mr pointer to 0 */
72
73     mr0 = 0; /* backup for later use */
74     DUART_SET_REG(REG_ADDR_MODE + chan->channel, mr0); /* MR0 */

```

```

75
76 /* set bits per character (8) and parity mode */
77 DUART_SET_REG(REG_ADDR_MODE + chan->channel,
78   0x3 | chan->parity); /* MR1 */
79
80 DUART_SET_REG(REG_ADDR_MODE + chan->channel,
81   chan->channel_mode | chan->stopbits); /* MR2 */
82
83 /*
84  * baudrate configuration
85  */
86 DUART_SET_REG(REG_ADDR_CLOCK_SELECT + chan->channel,
87   (duart_br[chan->baudrate][0] << 4) | /* recv */
88   duart_br[chan->baudrate][0] /* trans */
89 ); /* baud rate, same speed for transmitter and receiver */
90 DUART_SET_REG(REG_ADDR_COMMAND + chan->channel, 0xB0); /* mr pointer to 0 */
91 /* write MR0[2:0] */
92 DUART_SET_REG(REG_ADDR_MODE + chan->channel, mr0 | duart_br[chan->baudrate][2]);
93 /* set ACR[7] (ACR = 0x80 by default) */
94 DUART_SET_REG(REG_ADDR_ACR + chan->channel, (duart_br[chan->baudrate][1] << 8);
95
96 /* set the fifo */
97 chan->fifo = fifos[chan->channel / (DUART_CHANNEL_B - DUART_CHANNEL_A)];
98
99 /* put the channel active or inactive */
100 DUART_SET_REG(REG_ADDR_COMMAND + chan->channel, chan->enable);
101 }

```

### 6.2.3.5 void duart\_channel\_getattr (duart\_channel\_t \* chan)

Liest die Einstellungen vom Chip in die übergebene `duart_channel_t` (S.3) Struktur.

Aufgrund von chipseitigen Limitationen (manche Konfigurationsregister sind nur schreibbar aber nicht lesbar) können manche Werte nicht ermittelt werden. Diese werden so belassen wie sie übergeben wurden.

#### Parameter:

*chan* Zeiger auf die `duart_channel_t` (S.3) Struktur, welche benutzt werden soll

```

104 {
105   char c;
106   DUART_SET_REG(REG_ADDR_COMMAND + chan->channel, 0xB0); /* mr pointer to 0 */
107   c = DUART_GET_REG(REG_ADDR_MODE + chan->channel);
108   c = DUART_GET_REG(REG_ADDR_MODE + chan->channel);
109   chan->parity = (c & 0x1C) >> 2;
110   c = DUART_GET_REG(REG_ADDR_MODE + chan->channel);
111   chan->stopbits = (c & 0x0F);
112   chan->channel_mode = (c & 0xC0) >> 6;
113   chan->fifo = fifos[chan->channel / (DUART_CHANNEL_B - DUART_CHANNEL_A)];
114 }

```

### 6.2.3.6 duart\_channel\_t\* duart\_create\_channel ()

Legt eine `duart_channel_t` (S.3) Struktur an und füllt sie mit Standartwerten.

Eine `duart_channel_t`(S. 3) Struktur wird auf dem Heap angelegt und wird mit folgenden Werten gefüllt:

```
channel      = DUART_CHANNEL_A
baudrate    = DUART_B38400
channel_mode = DUART_CHANMODE_NORMAL
parity      = DUART_PARITY_MODE_NO
stopbits    = DUART_STOPBITS_1
```

Danach wird der Zeiger auf die Struktur zurückgegeben.

**Achtung:**

Eine mit `duart_create_channel()`(S. 20) angelegte Struktur muss mit `duart_destroy_channel()`(S. 21) wieder freigegeben werden!

**Rückgabe:**

Zeiger auf die erzeugte Struktur oder NULL wenn nicht erfolgreich

```
117 {
118     duart_channel_t *chan;
119
120     chan = (duart_channel_t *)malloc(sizeof(duart_channel_t));
121     if (!chan)
122         return NULL;
123
124     chan->channel      = DUART_CHANNEL_A;
125     chan->enable       = DUART_DISABLE_TX | DUART_DISABLE_RX;
126     chan->baudrate    = DUART_B38400;
127     chan->channel_mode = DUART_CHANMODE_NORMAL;
128     chan->parity      = DUART_PARITY_MODE_NO;
129     chan->stopbits    = DUART_STOPBITS_1;
130
131     return chan;
132 }
```

### 6.2.3.7 `void duart_destroy_channel(duart_channel_t *chan)`

Löscht die übergebene `duart_channel_t`(S. 3) Struktur vom Heap.

**Parameter:**

*chan* Zeiger auf die zu löschende `duart_channel_t`(S. 3) Struktur

```
135 {
136     free(chan);
137 }
```

### 6.2.3.8 `void duart_init_isr(char IntLvl, void(*) (void) isr)`

Initialisiert das Interrupt-Event-Handling.

Falls der Benutzer die Interruptbehandlung selbst übernehmen will, muss er seine ISR Funktion übergeben, ansonsten NULL.

**Parameter:**

*IntLvl* eingestellter Interruptlevel

*isr* Funktion die Aufgerufen werden soll, oder NULL

```

170 {
171
172   DUART_SET_REG(REG_ADDR_IVR, IntVec);
173   DUART_SET_REG(REG_ADDR_IMR, 0x22);
174
175   INTERRUPT_DISABLE;
176   IVT[IntVec] = duart_isr;
177   if (isr != NULL)
178       IVT[IntVec] = isr;
179   PFPAR |= 8; // Bit 3 im PFPAR setzen -> Eingang ist nun /IRQ3
180   //PEPAR &= ~4; // Bit 2 im PEPAR löschen -> /AVEC-Input deaktivieren
181   PEPAR |= 4; // Bit 2 im PEPAR setzen -> /AVEC-Input aktivieren
182   //IVT[27] = duart_isr; // AVEC 3 Vorsichtshalber
183   asm(" movel #IVT,%d0");
184   asm(" movec %d0,%vbr");
185   INTERRUPT_ENABLE;
186 }
```

**6.3 `duart_fifo.h`-Dateireferenz**

Implementierung einer Ringpuffer-FIFO für byteweises Speichern und Auslesen.

**Datenstrukturen**

- struct `duart_fifo_t`

*Diese Struktur hält alle wichtigen Information für jeweils eine FIFO.*

**Funktionen**

- `duart_fifo_t * duart_fifo_create (unsigned int size)`

*Erzeugt eine `duart_fifo_t`(S.4) Struktur und gibt den Zeiger darauf zurück.*

- `void duart_fifo_destroy (duart_fifo_t *fifo)`

*Löscht eine `duart_fifo_t`(S.4) Struktur vom Heap.*

- `void duart_fifo_write (duart_fifo_t *fifo, char c)`

*Schreibt ein Byte in die übergebene FIFO.*

- `char duart_fifo_read (duart_fifo_t *fifo)`

*Liest ein Byte aus der übergebenen FIFO (non-blocking).*

- `char duart_fifo_bread (duart_fifo_t *fifo)`

*Liest ein Byte aus der übergebenen FIFO (blocking).*

- char **duart\_fifo\_isempty** (duart\_fifo\_t \*fifo)  
*Gibt zurück ob der übergebene FIFO leer ist.*
- void **duart\_fifo\_reset** (duart\_fifo\_t \*fifo)  
*Setzt den übergebenen FIFO in den Anfangszustand zurück.*

### 6.3.1 Ausführliche Beschreibung

Implementierung einer Ringpuffer-FIFO für byteweises Speichern und Auslesen.

Diese FIFO ist in Form eines Ringpuffers implementiert und wird vom Treiber für die Pufferung der Daten, die vom jeweiligen Kanal gelesen werden, genutzt. Es werden Funktionen zum Erzeugen einer FIFO, zum Schreiben in die FIFO, zum Lesen aus der FIFO, zum Überprüfen des Füllstandes der FIFO und zum Zerstören einer bestehenden FIFO geboten. Weiterhin wird die Möglichkeit geboten Overflow's zu registrieren und eine FIFO auf einen definierten Anfangswert zurück zu setzen.

### 6.3.2 Dokumentation der Funktionen

#### 6.3.2.1 duart\_fifo\_t\* duart\_fifo\_create (unsigned int size)

Erzeugt eine **duart\_fifo\_t**(S. 4) Struktur und gibt den Zeiger darauf zurück.

Es wird eine **duart\_fifo\_t**(S. 4) Struktur auf dem Heap erzeugt, die Variablen initialisiert und anschließend der Zeiger auf die Struktur zurück gegeben.

#### Rückgabe:

Zeiger auf die erzeugte Struktur oder NULL wenn nicht erfolgreich

```

6 {
7   duart_fifo_t *fifo;
8
9   /* create struct on the heap */
10  fifo = (duart_fifo_t *)malloc(sizeof(duart_fifo_t));
11  if (!fifo)
12      return NULL;
13
14  fifo->size = size;
15  fifo->overflow = 0;
16
17  /* create buffer for the fifo */
18  fifo->buf = (char *)malloc(size);
19  if (!(fifo->buf))
20  {
21      /* if memory couldn't be allocated destroy the struct and return NULL */
22      duart_fifo_destroy(fifo);
23      return NULL;
24  }
25
26  /* set the buffer into a defined state (filled with 0x7e's (ascii for "~")) */
27  memset(fifo->buf, 0x7e, fifo->size); // tilde
28

```

```

29  /* setup the pointers */
30  fifo->next_elem = fifo->buf;
31  fifo->next_read = fifo->buf;
32
33  fifo->numbytes = 0;
34
35  return fifo;
36 }

```

### 6.3.2.2 void duart\_fifo\_destroy (duart\_fifo\_t \* fifo)

Löscht eine `duart_fifo_t`(S.4) Struktur vom Heap.

**Parameter:**

*fifo* Zeiger auf die zu löschende Struktur.

```

39 {
40  /* do nothing if the supplied pointer is NULL */
41  if (!fifo) return;
42
43  /* free the buffer ... */
44  free(fifo->buf);
45  /* ... and then the struct */
46  free(fifo);
47 }

```

### 6.3.2.3 void duart\_fifo\_write (duart\_fifo\_t \* fifo, char c)

Schreibt ein Byte in die übergebene FIFO.

Werden noch nicht gelesene Daten überschrieben wird die overflow-Variable auf eins (true) gesetzt.

**Parameter:**

*fifo* Zeiger auf die Struktur des FIFOs

*c* das zu schreibende Byte

```

50 {
51  /* write element and increase the number of bytes available in the buf */
52  *(fifo->next_elem) = c;
53  fifo->numbytes++;
54
55  /* if the pointers to the next element to read and the next element to write
56   * are equal check if a overflow has happened */
57  if (fifo->next_elem == fifo->next_read)
58      fifo->overflow = (fifo->numbytes > fifo->size);
59
60  /* if we have reached the end of the buffer make buf point to the beginning,
61   * else increase the pointer by one */
62  if (fifo->next_elem == fifo->buf + fifo->size - 1)
63      fifo->next_elem = fifo->buf;
64  else
65      fifo->next_elem++;
66 }

```

**6.3.2.4 char duart\_fifo\_read (duart\_fifo\_t \* fifo)**

Liest ein Byte aus der übergebenen FIFO (non-blocking).

Wird gelesen und die overflow-Variable ist aktiv (true), wird 0x21 (ascii für "!") zurückgegeben.

**Achtung:**

Wird gelesen und der FIFO ist leer, wird einfach das nächste Byte im Puffer ausgegeben und das nächste Byte was in den FIFO geschrieben wird geht verloren. Der Programmierer hat sich mit Hilfe von **duart\_fifo\_iseempty()**(S.26) zu vergewissern, das dieser Fall nicht eintritt.

```

69 {
70 /* read the next element from the buffer */
71 char c = *(fifo->next_read);
72
73 /* if a overflow has happened return a defined value (ascii for "!") */
74 if (fifo->overflow)
75     return 0x21; /* ausrufezeichen */
76
77 /* if the end of buffer has been reached point to the beginning,
78  * else increase by one */
79 if (fifo->next_read == fifo->buf + fifo->size - 1)
80     fifo->next_read = fifo->buf;
81 else
82     fifo->next_read++;
83
84 /* decrement numbytes by one */
85 fifo->numbytes--;
86
87 return c;
88 }

```

**6.3.2.5 char duart\_fifo\_bread (duart\_fifo\_t \* fifo)**

Liest ein Byte aus der übergebenen FIFO (blocking).

Mit dem Lesen wird solange gewartet, bis mindestens ein Byte im FIFO zu lesen ist. Ansonsten wie **duart\_fifo\_read()**(S.25).

**Parameter:**

*fifo* Zeiger auf die Struktur des FIFOs

**Rückgabe:**

das gelesene Byte oder 0x21

**Siehe auch:**

**duart\_fifo\_read()**(S.25)

```

91 {
92 /* wait until we can read a byte */
93 while (duart_fifo_iseempty(fifo));
94
95 /* read it and return */
96 return duart_fifo_read(fifo);
97 }

```

### 6.3.2.6 `char duart_fifo_isempty (duart_fifo_t * fifo)`

Gibt zurück ob der übergebene FIFO leer ist.

**Parameter:**

*fifo* Zeiger auf die Struktur des FIFOs

**Rückgabe:**

true wenn leer, false wenn gefüllt

```
100 {  
101     return (fifo->numbytes == 0);  
102 }
```

## 6.4 `duart_registers.h`-Dateireferenz

Die Adressen zu den Registern des Chipsatzes werden festgelegt.

**Makrodefinitionen**

- `#define REG_ADDR_MODE 0x400`  
*Adresse vom Mode-Register.*
- `#define REG_ADDR_STATUS 0x402`  
*Adresse vom Status-Register.*
- `#define REG_ADDR_CLOCK_SELECT 0x402`  
*Adresse vom Clock-Select-Register.*
- `#define REG_ADDR_COMMAND 0x404`  
*Adresse vom Command-Register.*
- `#define REG_ADDR_RXFIFO 0x406`  
*Adresse vom Empfangs-FIFO-Register.*
- `#define REG_ADDR_TXFIFO 0x406`  
*Adresse vom Sender-FIFO-Register.*
- `#define REG_ADDR_IPCR 0x408`  
*Adresse vom Input Port Change Register.*
- `#define REG_ADDR_ACR 0x408`  
*Adresse vom Auxillary Control-Register.*
- `#define REG_ADDR_ISR 0x40A`  
*Adresse vom Interrupt Status-Register.*

- #define **REG\_ADDR\_IMR** 0x40A  
*Adresse vom Interrupt Mask-Register.*
- #define **REG\_ADDR\_CTU** 0x40C  
*Adresse vom Counter Timer Upper-Register.*
- #define **REG\_ADDR\_CTPU** 0x40C  
*Adresse vom Counter Timer Upper Preset-Register.*
- #define **REG\_ADDR\_CTL** 0x40E  
*Adresse vom Counter Timer Lower-Register.*
- #define **REG\_ADDR\_CTPL** 0x40E  
*Adresse vom Counter Timer Lower Preset-Register.*
- #define **REG\_ADDR\_IVR** 0x418  
*Adresse vom Interrupt Vector-Register.*
- #define **REG\_ADDR\_IPR** 0x41A  
*Adresse vom Input Port-Register.*
- #define **REG\_ADDR\_STARTCC** 0x41C  
*Adresse vom Start Counter Command-Register.*
- #define **REG\_ADDR\_SOPR** 0x41C  
*Adresse vom Set Output Port Bits Command-Register.*
- #define **REG\_ADDR\_STOPCC** 0x41E  
*Adresse vom Stop Counter Command-Register.*
- #define **REG\_ADDR\_ROPR** 0x41E  
*Adresse vom Reset Output Port Bits Command-Register.*

#### 6.4.1 Ausführliche Beschreibung

Die Adressen zu den Registern des Chipsatzes werden festgelegt.

Da die Register memory-mapped sind, sind die Werte hier Offsets zur Basis-Adresse, die der Funktion `duart_init()`(S. 18) übergeben wird.

## 7 Duart Driver Zusätzliche Informationen

### 7.1 Liste der zu erledigenden Dinge

**Global** `duart_fifo_write(S. 24)(duart_fifo_t(S. 4) *fifo, char c)`  
numbytes zurücksetzen wenn overflow stattgefunden hat

## Index

duart\_baudrates.h, 5  
DUART\_CHANMODE\_AUTO\_-  
ECHO  
duart\_driver.h, 17  
DUART\_CHANMODE\_-  
LOCAL\_LOOP  
duart\_driver.h, 17  
DUART\_CHANMODE\_-  
NORMAL  
duart\_driver.h, 17  
DUART\_CHANMODE\_-  
REMOTE\_LOOP  
duart\_driver.h, 17  
DUART\_CHANNEL\_A  
duart\_driver.h, 11  
DUART\_CHANNEL\_B  
duart\_driver.h, 11  
duart\_channel\_getattr  
duart\_driver.h, 20  
duart\_channel\_setattr  
duart\_driver.h, 19  
duart\_channel\_t, 2  
parity, 3  
duart\_create\_channel  
duart\_driver.h, 20  
duart\_destroy\_channel  
duart\_driver.h, 21  
DUART\_DISABLE\_RX  
duart\_driver.h, 12  
DUART\_DISABLE\_TX  
duart\_driver.h, 11  
duart\_driver.h, 6  
DUART\_CHANMODE\_-  
AUTO\_ECHO, 17  
DUART\_CHANMODE\_-  
LOCAL\_LOOP, 17  
DUART\_CHANMODE\_-  
NORMAL, 17  
DUART\_CHANMODE\_-  
REMOTE\_LOOP, 17  
DUART\_CHANNEL\_A, 11  
DUART\_CHANNEL\_B, 11  
duart\_channel\_getattr, 20  
duart\_channel\_setattr, 19  
duart\_create\_channel, 20  
duart\_destroy\_channel, 21  
DUART\_DISABLE\_RX, 12  
DUART\_DISABLE\_TX, 11  
DUART\_ENABLE\_RX, 11  
DUART\_ENABLE\_TX, 11  
DUART\_GET\_REG, 11  
duart\_getchar, 19  
duart\_init, 18  
duart\_init\_isr, 21  
DUART\_PARITY\_EVEN, 12  
DUART\_PARITY\_MODE\_-  
FORCE, 12  
DUART\_PARITY\_MODE\_-  
MULTIDROP, 13  
DUART\_PARITY\_MODE\_-  
NO, 12  
DUART\_PARITY\_MODE\_-  
WITH, 12  
DUART\_PARITY\_ODD, 12  
DUART\_SET\_REG, 11  
DUART\_STOPBITS5\_1\_-  
063, 15  
DUART\_STOPBITS5\_1\_-  
125, 16  
DUART\_STOPBITS5\_1\_-  
188, 16  
DUART\_STOPBITS5\_1\_-  
250, 16  
DUART\_STOPBITS5\_1\_-  
313, 16  
DUART\_STOPBITS5\_1\_-  
375, 16  
DUART\_STOPBITS5\_1\_-  
438, 16  
DUART\_STOPBITS5\_1\_5,  
17  
DUART\_STOPBITS\_0\_563,  
13  
DUART\_STOPBITS\_0\_625,  
13  
DUART\_STOPBITS\_0\_688,  
13  
DUART\_STOPBITS\_0\_750,  
13  
DUART\_STOPBITS\_0\_813,  
13  
DUART\_STOPBITS\_0\_875,  
14  
DUART\_STOPBITS\_0\_938,

- 14
- DUART\_STOPBITS\_1, 14
- DUART\_STOPBITS\_1\_563, 14
- DUART\_STOPBITS\_1\_625, 14
- DUART\_STOPBITS\_1\_688, 14
- DUART\_STOPBITS\_1\_750, 15
- DUART\_STOPBITS\_1\_813, 15
- DUART\_STOPBITS\_1\_875, 15
- DUART\_STOPBITS\_1\_938, 15
- DUART\_STOPBITS\_2, 15
- duart\_write, 18
- DUART\_ENABLE\_RX
  - duart\_driver.h, 11
- DUART\_ENABLE\_TX
  - duart\_driver.h, 11
- duart\_fifo.h, 22
  - duart\_fifo\_bread, 25
  - duart\_fifo\_create, 23
  - duart\_fifo\_destroy, 23
  - duart\_fifo\_isempty, 25
  - duart\_fifo\_read, 24
  - duart\_fifo\_write, 24
- duart\_fifo\_bread
  - duart\_fifo.h, 25
- duart\_fifo\_create
  - duart\_fifo.h, 23
- duart\_fifo\_destroy
  - duart\_fifo.h, 23
- duart\_fifo\_isempty
  - duart\_fifo.h, 25
- duart\_fifo\_read
  - duart\_fifo.h, 24
- duart\_fifo\_t, 3
  - overflow, 4
- duart\_fifo\_write
  - duart\_fifo.h, 24
- DUART\_GET\_REG
  - duart\_driver.h, 11
- duart\_getchar
  - duart\_driver.h, 19
- duart\_init
  - duart\_driver.h, 18
- duart\_init\_isr
  - duart\_driver.h, 21
- DUART\_PARITY\_EVEN
  - duart\_driver.h, 12
- DUART\_PARITY\_MODE\_-FORCE
  - duart\_driver.h, 12
- DUART\_PARITY\_MODE\_-MULTIDROP
  - duart\_driver.h, 13
- DUART\_PARITY\_MODE\_NO
  - duart\_driver.h, 12
- DUART\_PARITY\_MODE\_-WITH
  - duart\_driver.h, 12
- DUART\_PARITY\_ODD
  - duart\_driver.h, 12
- duart\_registers.h, 26
- DUART\_SET\_REG
  - duart\_driver.h, 11
- DUART\_STOPBITS5\_1\_063
  - duart\_driver.h, 15
- DUART\_STOPBITS5\_1\_125
  - duart\_driver.h, 16
- DUART\_STOPBITS5\_1\_188
  - duart\_driver.h, 16
- DUART\_STOPBITS5\_1\_250
  - duart\_driver.h, 16
- DUART\_STOPBITS5\_1\_313
  - duart\_driver.h, 16
- DUART\_STOPBITS5\_1\_375
  - duart\_driver.h, 16
- DUART\_STOPBITS5\_1\_438
  - duart\_driver.h, 16
- DUART\_STOPBITS5\_1\_5
  - duart\_driver.h, 17
- DUART\_STOPBITS\_0\_563
  - duart\_driver.h, 13
- DUART\_STOPBITS\_0\_625
  - duart\_driver.h, 13
- DUART\_STOPBITS\_0\_688
  - duart\_driver.h, 13
- DUART\_STOPBITS\_0\_750
  - duart\_driver.h, 13
- DUART\_STOPBITS\_0\_813
  - duart\_driver.h, 13
- DUART\_STOPBITS\_0\_875
  - duart\_driver.h, 14
- DUART\_STOPBITS\_0\_938
  - duart\_driver.h, 14
- DUART\_STOPBITS\_1

---

duart\_driver.h, 14  
DUART\_STOPBITS\_1\_563  
duart\_driver.h, 14  
DUART\_STOPBITS\_1\_625  
duart\_driver.h, 14  
DUART\_STOPBITS\_1\_688  
duart\_driver.h, 14  
DUART\_STOPBITS\_1\_750  
duart\_driver.h, 15  
DUART\_STOPBITS\_1\_813  
duart\_driver.h, 15  
DUART\_STOPBITS\_1\_875  
duart\_driver.h, 15  
DUART\_STOPBITS\_1\_938  
duart\_driver.h, 15  
DUART\_STOPBITS\_2  
duart\_driver.h, 15  
duart\_write  
duart\_driver.h, 18

overflow  
duart\_fifo\_t, 4

parity  
duart\_channel\_t, 3