AM335x Software UART Architecture

Software Architecture Document
Read Me

ABOUT THIS MANUAL

This Quick Start Guide provides developers a quick start to get the Software Development Platform AM335x up and running as soon as possible. It should not be the only source of information for the developer. Refer to the documentation specified in the “Related Documentation” section below.

RELATED DOCUMENTATION

IF YOU NEED ASSISTANCE

Contact your TI sales representative.

FCC WARNING

This equipment is intended for use in a laboratory test environment only. It generates uses and can radiate radio frequency energy and has not been tested for compliance with the limits of computing devices pursuant J of part 15 of FCC rules, which are designed to provide reasonable protection against radio frequency interference. Operation of this equipment in their own expense will be required to take whatever measures may be required to correct this interference.
Table of Contents

AM335X SOFTWARE UART ARCHITECTURE ................................................................................. 1

READ ME .......................................................................................................................... 2

ABOUT THIS MANUAL ....................................................................................................... 2
RELATED DOCUMENTATION .......................................................................................... 2
IF YOU NEED ASSISTANCE .............................................................................................. 2
FCC WARNING ................................................................................................................ 2

1. INTRODUCTION ............................................................................................................. 6

1.1 PURPOSE .................................................................................................................. 6
1.2 SCOPE ...................................................................................................................... 6
1.3 TERMS / ACRONYMS / ABBREVIATIONS ............................................................... 6
1.4 BIBLIOGRAPHY ....................................................................................................... 6

2. DESIGN CONSIDERATIONS .......................................................................................... 7

2.1. ASSUMPTIONS ....................................................................................................... 7
2.2. CONSTRAINTS ......................................................................................................... 7
2.3. DEPENDENCIES ................................................................................................... 7

3. SYSTEM OVERVIEW ...................................................................................................... 8

4. SYSTEM ARCHITECTURE ............................................................................................. 9

4.1 SYSTEM BLOCK DIAGRAM ....................................................................................... 9

THE AM335X PRU EXTENSION DAUGHTERBOARD WILL PLUG ON THE ................. 9

4.2 FUNCTIONAL BLOCK DESCRIPTION ..................................................................... 12
  4.2.2 Data Format ....................................................................................................... 13
4.3 FUNCTIONAL VIEW .................................................................................................. 14
  4.3.1 Transmission ...................................................................................................... 14
  4.3.2 Reception .......................................................................................................... 17
4.4 PRU ASSEMBLY FLOW ........................................................................................... 19
4.5 SYSTEM RESOURCE ............................................................................................... 21
  4.5.1 Memory .............................................................................................................. 21
  4.5.2 Interrupts ........................................................................................................... 22

5. IMPLEMENTATION DETAILS ....................................................................................... 24

  5.1 Soft UART Registers per PRU .................................................................................. 24
  5.1.1 Channel Control and Configuration Registers .................................................. 27
  5.1.2 Transmit/Receive Registers ............................................................................... 30
  5.2 UART Global Registers .......................................................................................... 33
  5.2.1 UART Interrupt Mask Register ........................................................................ 33
  5.2.1 UART Interrupt Status Register ....................................................................... 34

6 SOFTWARE DESIGN – API LIST .................................................................................. 35

  DATA STRUCTURES ....................................................................................................... 35
6.1.1. suart_config ................................................................. 35
6.1.2. suart_struct_handle ...................................................... 36
6.1.3. tSuartInfo ................................................................. 37
6.2 Function ........................................................................ 38
6.2.1 pru_softuart_init ............................................................ 38
6.2.2 pru_set_fifo_timeout ....................................................... 39
6.2.3 pru_mcasp_deinit ............................................................ 39
6.2.4 pru_softuart_deinit ......................................................... 39
6.2.5 PRU_SoftUart_Open ......................................................... 40
6.2.6 pru_softuart_close .......................................................... 40
6.2.7 PRU_SoftUart_SetBaud ..................................................... 41
6.2.8 PRU_SoftUart_SetParity .................................................... 41
6.2.9 PRU_SoftUart_SetDataBits ................................................ 42
6.2.10 pru_softuart_setconfig ................................................... 42
6.2.11 pru_soft uart_getTxDataLen ............................................. 43
6.2.12 pru_softuart_getRxDataLen ............................................. 43
6.2.13 PRU_SoftUart_GetConfig ................................................ 44
6.2.14 pru_softuart_write ......................................................... 44
6.2.15 pru_softuart_read .......................................................... 45
6.2.16 pru_softuart_read_data .................................................... 45
6.2.17 pru_softuart_getTxStatus ................................................. 46
6.2.18 pru_softuart_clrTxStatus ............................................... 47
6.2.19 pru_softuart_getRxStatus ............................................... 47
6.2.20 pru_softuart_clrRxFifo .................................................... 47
6.2.21 pru_softuart_clrRxStatus ............................................... 48
6.2.22 pru_softuart_get_isrstatus .............................................. 48
6.2.23 pru_intr_clr_isrstatus ................................................... 49
6.2.24 suart_pru_to_host_intr_enable ....................................... 49
6.2.25 suart_intr_setmask ....................................................... 50
6.2.26 suart_intr_clrmask ....................................................... 50

APPENDIX A .......................................................................... 52
Firmware Flow Diagrams ......................................................... 52
## Revision History

<table>
<thead>
<tr>
<th>REV</th>
<th>DATE</th>
<th>CHANGES</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>27-Aug.-2012</td>
<td>AM335x Initial Release</td>
</tr>
</tbody>
</table>
1. Introduction

This document describes the software implementation of Soft Universal Asynchronous Receiver/Transmitter (UART) peripheral emulation using the Multichannel audio serial ports on AM335x.

1.1 Purpose

AM335x can support up to 4 Soft-UARTs with on-board peripherals, 2 using McASP0 and another 2 using McASP1. The daughterboard limits the number of UARTS to 3 due to pinmux constraints with the AM335x GP EVM baseboard. The document provides a brief overview of the software architecture and implementation of Soft-UARTs using the PRU and McASP on AM335x board along with AM335x PRU daughterboard.

1.2 Scope

This document explains only the software part of UART implementation. This document limits its scope to the illustration of the software architecture and API description of the soft-UART implementation. This document neither explains any hardware details nor does it provide detailed implementation of PRU architecture. For more information on the PRU Sub-system, please refer to on-line documentation provided by TI.

1.3 Terms / Acronyms / Abbreviations

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM</td>
<td>Advanced RISC Machine</td>
</tr>
<tr>
<td>CCS</td>
<td>Code Composer Studio</td>
</tr>
<tr>
<td>DSP</td>
<td>Digital Signal Processing</td>
</tr>
<tr>
<td>DSP/BIOS</td>
<td>Texas Instruments Real-time Operating System Kernel</td>
</tr>
<tr>
<td>EVM</td>
<td>Evaluation Module</td>
</tr>
<tr>
<td>McASP</td>
<td>Multichannel Audio Serial Port</td>
</tr>
<tr>
<td>OMAP</td>
<td>Open Multimedia Applications Platform</td>
</tr>
<tr>
<td>PRUSS</td>
<td>Programmable Real-Time Unit SubSystem</td>
</tr>
<tr>
<td>RISC</td>
<td>Reduced Instruction Set Computer</td>
</tr>
<tr>
<td>UART</td>
<td>Universal Asynchronous Receiver/Transmitter</td>
</tr>
</tbody>
</table>

1.4 Bibliography

- AM335x PRU Reference Guide.pdf
2. Design Considerations

- The Emulation is not designed to be generic but specific to cater the requirement of AM335x.
- Soft UART emulation using McASP to communicate with UART.
- Design transmits or receives data based on user input of baud rate, bits per character.

2.1. Assumptions

- AM335x SDK tool chain (arm-arago-linux-gnueabi- ) is used to build PRU UART Linux driver on the ARM side.
- PRU Assembler is used to build PRU UART emulation source on PRU side.
- Open source Linux kernel 3.2 is used as base for initial development.

2.2. Constraints

The following features are supported in the soft UART
- Buffer and shift register capability.
- Baud clock Divisor registers to configure clocks using internal clocks
- Interrupts Transmit Data Ready and Receive Data Ready.
- Line control for Error Checks.
- FIFO modes to reduce system overhead.
- Even, odd, and none parity detection and generation.

The following features are not support in soft UART
- Receiver time out interrupt and line control interrupt.
- Hardware handshaking signals: UART_CTS and UART_RTS.

2.3. Dependencies

- Availability of AM335x GP EVM and PRU Daughterboard and associated software.
- Availability of PRU Assembler.
- Availability of boot loader and Linux kernel source.
3. System Overview

The block diagram gives an overview of the system.

Figure 0-1: Block diagram overview
4. System Architecture

4.1 System Block diagram

The AM335x PRU Extension daughterboard will plug on the AM335x GP EVM. The daughterboard will have three soft UART ports and four direct UARTs as shown below.

Figure 4-1: Hardware Architecture of PRU Expansion Board
Figure 4-2: Software Architecture of Soft-UART implementation

Figure 4-2 illustrates the various blocks involved in the Soft-UART implementation.

1. **Linux Serial Driver:** - This layer is the standard Terminal based tty driver for Linux Operating system. This layer uses the services provided by the Soft-UART API layer to provide the standard Terminal driver for Linux. This runs in kernel space on ARM CPU.

2. **Shared RAM:** - This is the internal Shared RAM on AM335x processor. This is primarily used for maintaining the FIFO buffers. Although the FIFO can reside in any type of memory (External DDR, SRAM or Internal RAM), Shared RAM is chosen as default to minimize the access time and thereby improve the performance.

3. **PRUSS:** - This includes the following
   a. **PRU Intc:** - PRU interrupt controller for handling inter processor communications
b. PRU RAM[0, 1]: - This is RAM of each PRU. This RAM is used for storing the Instruction code for PRU and any other data used for assembly code execution.

c. PRU[0,1]: - This is the RISC micro-controller executing the firmware implementing the soft-UART.

4. McASPx: - This implementation uses the McASP0 or McASP1 peripheral port for emulating the serial port. The McASP is used as a independent engine for shifting the data out to the Transmit serializer (for Tx operations) and also shifting the data into the shift registers from the Receive serializers (for Rx operations). For more detailed information please refer to subsequent sections.
4.2 Functional Block Description

Figure 4-3: Functional block diagram
The AM335x has 2 MCASP, which can be assigned to either PRU (by default, MCASP1 is assigned to PRU0 and MCASP0 is assigned to PRU1). Each McASP has 4 serializers, which are equipped with a buffer and a shift register for transmission and receiving. We can configure each serializer either transmitter or receiver. Depending upon transmit and/or receive, UART can be FULL UART, HALF UART.

- Up to 2 FULL UART on single PRU, both TX and RX constitute FULL UART.
- Up to 4 Full Soft-UARTs on both PRU.

In the software implementation of the UART using the McASP, the data transfers will be configured in the Time Division Multiplexing (TDM) Mode. The UART format data will be transmitted or received during the time slots of the TDM frame. The timing requirements of the UART protocol will be achieved by dividing the internal clock of the McASP. This eliminates the requirement of using any of the on-chip timers or implementing a timer on the software side. One of the McASP serializers will be configured to perform the transmit function while another McASP serializers would be configured to act as the receiver section of the UART. The PRU would act as the controlling device, which formats the data and handles events.

### 4.2.2 Data Format

The soft UART would have the capability to transmit in the following formats:

| 1 START bit | data bits (6, 5, 6, 7, 8, 9, 10, 11, 12) | 1 STOP bit |

The UART receives in the following format:

| 1 START bit | data bits (6, 5, 6, 7, 8, 9, 10, 11, 12) | 1 STOP bit |

Transmit/Receive for 7-bit data, 1 STOP bit

| D0 | D1 | D2 | D3 | D4 | D5 | D6 | STOP1 |

Transmit/Receive for 8-bit data, 1 STOP bit

| D0 | D1 | D2 | D3 | D4 | D5 | D6 | D7 | STOP1 |

Minimum size of transmit frame format is

8 bits = 1 start bit + 6 data bits + 1 stop bit

Maximum size of transmit frame format is

14 bits = 1 start bit + 12 data bits + 1 stop bit

The mentioned frame format size is size of the transmit frame format, not the actual size of the data buffer.
4.3 Functional View

This section provides a brief overview of transmit and receive functionalities.

4.3.1 Transmission

The following section explains the roles of ARM and PRU with respect to transmission.
4.3.1.1 Control flow diagram

This section illustrates the control flow for Transmit operation in single PRU mode.

Figure 4-4: Initialization of Transmit & IDLE State

Figure 4-5 Transmission control flow
4.3.1.2 ARM

- Configures McASP serializers to operate as transmitter
- Sets up the parameters of formatting and modes for data transfer
- Enables interrupts to and from the PRU and the McASP

Figure 4-6: Software block diagram for ARM Linux
4.3.1.3 PRU

- Acts as formatter during the transmission mode
- Polls for SU_THR_EMPT interrupt from the ARM (to send next data to be transmitted)
- Convert data into formatted data of the size of the time slot programmed in the McASP TDM mode
- On interrupt, it writes the formatted data to the SU_THR on McASP port
- In absence of any transmit data, it will generate words representing the idle state.

We can map any of the serializers to any Soft UART channels and can be configurable either transmit or receive.

**NOTE:**

In UART expansion board, the serializers are fixed to specific UART as shown in the table below.

<table>
<thead>
<tr>
<th>McASP Serializers</th>
<th>UART</th>
</tr>
</thead>
<tbody>
<tr>
<td>McASP0_AXR2</td>
<td>SoftUART1_TXD</td>
</tr>
<tr>
<td>McASP0_AXR3</td>
<td>SoftUART2_TXD</td>
</tr>
<tr>
<td>McASP1_AXR1</td>
<td>SoftUART3_TXD</td>
</tr>
</tbody>
</table>

4.3.2 Reception

The following section explains the roles of ARM, DSP and PRU with respect to reception.

4.3.2.1 Control flow diagram

This section illustrates the control flow for Receive operation in both Single PRU and Both PRU modes.

![Control flow diagram](image)

Figure 4-7: Receive operation initialization control flow
4.3.2.2 ARM

- Configures McASP serializers to operate as receivers
- Sets up the parameters of formatting and modes for data transfer
- Enables interrupts to and from the PRU and the McASP

4.3.2.3 PRU

- Acts as translator during the reception mode
- Polls for receive data ready (RDA_INT) interrupt from the McASP serializers
- When it receives the interrupt, it reads the over sampled data from the SU_RBR
- In order to determine start (high to low transition) bit, the PRU scans the bits of the over sampled data to determine the location of transition.
- It registers the bit location of the transition and always polls that bit of the subsequent received over sampled data to determine the received bit.

We can map any of the serializers to any Soft UART channels and can be configurable either transmit or receive.

NOTE:
In UART expansion board, the serializers are fixed to specific UART as shown in the table below

<table>
<thead>
<tr>
<th>McASP Serializers</th>
<th>UART</th>
</tr>
</thead>
<tbody>
<tr>
<td>McASP0_AXR0</td>
<td>SoftUART1_RXD</td>
</tr>
<tr>
<td>McASP0_AXR1</td>
<td>SoftUART2_RXD</td>
</tr>
<tr>
<td>McASP1_AXR0</td>
<td>SoftUART3_RXD</td>
</tr>
</tbody>
</table>
4.4 PRU Assembly flow

Each PRU runs the firmware implementing the soft-UART using the McASP. In single PRU mode, the PRU0 runs the firmware. When both PRUs are used for additional Soft UARTs, both PRU0 and PRU1 execute the same firmware code. The flow charts provided in this section gives an overview of the execution sequence of the firmware. Please refer to the Appendix A for detailed flowcharts of operation.

The PRU firmware basically consists of a CORE_LOOP, two system (ARM) event handlers and two McASP event handlers. The PRU keeps spinning on the CORE_LOOP until a system event is generated by the McASP or ARM. The four handlers are:

- **TxServiceReqHndlr**: This event is generated from the ARM to the PRU. The PRU on reception of this event copies the data from the shared RAM into the PRU RAM and processes it.

- **TxIntrHndlr**: This event is generated from the McASP to the PRU. The PRU on reception of this event pre-scales and copies the data into the McASP transmit buffers. When transmission completes, the PRU interrupts the ARM back.

- **RxServiceReqHndlr**: This event is generated from the ARM/DSP to the PRU. The PRU on reception of this event registers the ARM request for receiving data (max fifo size 16 chars).

- **RxIntrHndlr**: This event is generated from the McASP to the PRU. The PRU on reception of this event processes the received data and copies it into shared ram. When reception completes, the PRU interrupts the ARM back.
Figure 4-9: Main loop of firmware operation
### 4.5 System Resource.

#### 4.5.1 Memory

The following table has the data RAM of PRU memory map table for soft UART:

<table>
<thead>
<tr>
<th>Region</th>
<th>Description</th>
<th>Start Address</th>
<th>End Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data RAM 0 (PRU0)</td>
<td>Channel0</td>
<td>0x4A300000</td>
<td>0x4A30000F</td>
</tr>
<tr>
<td></td>
<td>Channel1</td>
<td>0x4A300010</td>
<td>0x4A30001F</td>
</tr>
<tr>
<td></td>
<td>Channel2</td>
<td>0x4A300020</td>
<td>0x4A30002F</td>
</tr>
<tr>
<td></td>
<td>Channel3</td>
<td>0x4A300030</td>
<td>0x4A30003F</td>
</tr>
<tr>
<td></td>
<td>Global Interrupt Mask Register</td>
<td>0x4A300080</td>
<td>0x4A300081</td>
</tr>
<tr>
<td></td>
<td>Global Interrupt Status Register</td>
<td>0x4A300082</td>
<td>0x4A300083</td>
</tr>
<tr>
<td></td>
<td>PRU ID Register</td>
<td>0x4A300084</td>
<td>0x4A300084</td>
</tr>
<tr>
<td></td>
<td>PRU RX/TX Mode Register</td>
<td>0x4A300085</td>
<td>0x4A300085</td>
</tr>
<tr>
<td></td>
<td>Unused</td>
<td>0x4A300086</td>
<td>0x4A300086</td>
</tr>
<tr>
<td></td>
<td>Max RX Timeout Retries Count</td>
<td>0x4A300088</td>
<td>0x4A300089</td>
</tr>
<tr>
<td></td>
<td>Context Info</td>
<td>0x4A300090</td>
<td>0x4A3001FF</td>
</tr>
<tr>
<td>Data RAM 1 (PRU1)</td>
<td>Channel0</td>
<td>0x4A302000</td>
<td>0x4A30200F</td>
</tr>
<tr>
<td></td>
<td>Channel1</td>
<td>0x4A302010</td>
<td>0x4A30201F</td>
</tr>
<tr>
<td></td>
<td>Channel2</td>
<td>0x4A302020</td>
<td>0x4A30202F</td>
</tr>
<tr>
<td></td>
<td>Channel3</td>
<td>0x4A302030</td>
<td>0x4A30203F</td>
</tr>
<tr>
<td></td>
<td>Global Interrupt Mask Register</td>
<td>0x4A302080</td>
<td>0x4A302081</td>
</tr>
<tr>
<td></td>
<td>Global Interrupt Status Register</td>
<td>0x4A302082</td>
<td>0x4A302083</td>
</tr>
<tr>
<td></td>
<td>PRU ID Register</td>
<td>0x4A302084</td>
<td>0x4A302084</td>
</tr>
<tr>
<td>PRU RX/TX Mode Register</td>
<td>0x4A302085</td>
<td>0x4A302085</td>
<td></td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------</td>
<td>-----------</td>
<td></td>
</tr>
<tr>
<td>Unused</td>
<td>0x4A302086</td>
<td>0x4A302086</td>
<td></td>
</tr>
<tr>
<td>Max RX Timeout Retries Count</td>
<td>0x4A302088</td>
<td>0x4A302089</td>
<td></td>
</tr>
<tr>
<td>Unused</td>
<td>0x4A30208A</td>
<td>0x4A30208F</td>
<td></td>
</tr>
<tr>
<td>Context Info</td>
<td>0x4A302090</td>
<td>0x4A3021FF</td>
<td></td>
</tr>
</tbody>
</table>

**Table 4-1: Memory layout**

The registers are maintained on per channel basis. Each PRU contain 4 channels.

### 4.5.2 Interrupts

The Soft-UART uses System events to communicate across Host CPU and the PRU for notifying the events. The below table provides a summary of the system events; their corresponding mapping to the channel number and its usage.

<table>
<thead>
<tr>
<th>System Event Number</th>
<th>Host channel mapping</th>
<th>DSP Interrupt Number</th>
<th>ARM INTC Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>0</td>
<td>0</td>
<td>NA</td>
<td>Host CPU to PRU0 interrupt for communicating any service request</td>
</tr>
<tr>
<td>17</td>
<td>1</td>
<td>1</td>
<td>NA</td>
<td>Host CPU to PRU1 interrupt for communicating any service request</td>
</tr>
<tr>
<td>20</td>
<td>2</td>
<td>2</td>
<td>20</td>
<td>Soft-UART 2 Tx channel events</td>
</tr>
<tr>
<td>21</td>
<td>2</td>
<td>2</td>
<td>20</td>
<td>Soft-UART 2 Rx Channel Events</td>
</tr>
<tr>
<td>22</td>
<td>3</td>
<td>3</td>
<td>21</td>
<td>Soft-UART 3 Tx channel events</td>
</tr>
<tr>
<td>23</td>
<td>3</td>
<td>3</td>
<td>21</td>
<td>Soft-UART 3 Rx Channel Events</td>
</tr>
<tr>
<td>24</td>
<td>4</td>
<td>4</td>
<td>24</td>
<td>Soft-UART 0 Tx channel events</td>
</tr>
<tr>
<td>25</td>
<td>4</td>
<td>4</td>
<td>24</td>
<td>Soft-UART 0 Rx Channel Events</td>
</tr>
<tr>
<td>26</td>
<td>5</td>
<td>5</td>
<td>25</td>
<td>Soft-UART 1 Tx channel events</td>
</tr>
<tr>
<td>27</td>
<td>5</td>
<td>5</td>
<td>25</td>
<td>Soft-UART 1 Rx Channel Events</td>
</tr>
<tr>
<td>33</td>
<td>0</td>
<td>0</td>
<td>NA</td>
<td>McASP1 to PRU0 interrupt</td>
</tr>
<tr>
<td>34</td>
<td>0</td>
<td>0</td>
<td>NA</td>
<td>McASP1 to PRU0 interrupt</td>
</tr>
<tr>
<td>54</td>
<td>1</td>
<td>1</td>
<td>NA</td>
<td>McASP0 to PRU1 interrupt</td>
</tr>
<tr>
<td>55</td>
<td>1</td>
<td>1</td>
<td>NA</td>
<td>McASP0 to PRU1 interrupt</td>
</tr>
</tbody>
</table>
Table 4-2 System Events corresponding to Soft-UART
5. Implementation details

5.1 Soft UART Registers per PRU

Each PRU contains the 4 Soft UART channels and each serializer can be mapped to any one of these channels. The mapping information will be in board specific file named `am335x_suart_board.h`. Each channel can be configurable to either transmit or receive.

Full Soft UART or Half Soft UART can be achieved by configuring in the file `am335x_suart_board.h`. When user opens an instance for particular Soft UART, the instance will be opened with FULL UART or Half UART depending upon the configuration file. The file `am335x_suart_board.h` will need to be modified depending upon the hardware configuration.

Each channel contains control register, configuration register and Transmit/Receive registers as shown below.
The channels offset for each PRU is as shown below

<table>
<thead>
<tr>
<th>Soft UART Channel</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel 0</td>
<td>0x00</td>
</tr>
<tr>
<td>Channel 1</td>
<td>0x10</td>
</tr>
<tr>
<td>Channel 2</td>
<td>0x20</td>
</tr>
<tr>
<td>Channel 3</td>
<td>0x30</td>
</tr>
</tbody>
</table>

For each Channel contains the registers as shown below

<table>
<thead>
<tr>
<th>Section</th>
<th>Register</th>
<th>Start Address</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRU0</td>
<td>CHn_Ctrl</td>
<td>0x4A300000 + (16 * n)</td>
<td>16 bits</td>
</tr>
<tr>
<td></td>
<td>CHn_Config1</td>
<td>0x4A300002 + (16 * n)</td>
<td>16 bits</td>
</tr>
<tr>
<td></td>
<td>CHn_Config2</td>
<td>0x4A300004 + (16 * n)</td>
<td>16 bits</td>
</tr>
<tr>
<td></td>
<td>CHn_TXRXStatus</td>
<td>0x4A300006 + (16 * n)</td>
<td>16 bits</td>
</tr>
<tr>
<td></td>
<td>CHn_TXRXData</td>
<td>0x4A300008 + (16 * n)</td>
<td>32 bits</td>
</tr>
<tr>
<td>PRU1</td>
<td>CHn_Ctrl</td>
<td>0x4A302000 + (16 * n)</td>
<td>16 bits</td>
</tr>
<tr>
<td></td>
<td>CHn_Config1</td>
<td>0x4A302002 + (16 * n)</td>
<td>16 bits</td>
</tr>
<tr>
<td></td>
<td>CHn_Config2</td>
<td>0x4A302004 + (16 * n)</td>
<td>16 bits</td>
</tr>
<tr>
<td></td>
<td>CHn_TXRXStatus</td>
<td>0x4A302006 + (16 * n)</td>
<td>16 bits</td>
</tr>
<tr>
<td></td>
<td>CHn_TXRXData</td>
<td>0x4A302008 + (16 * n)</td>
<td>32 bits</td>
</tr>
</tbody>
</table>

Where ‘n’ is the Soft UART channel number.
### 5.1.1 Channel Control and Configuration Registers

#### 5.1.1.1 UART Channel Control Registers (CHn_Ctrl)

This register gives the bit descriptions for each channel. There are 8 channels for both PRUs. PRU0 has 4 channels and PRU1 has 4 channels. Each channel can be mapped to any one of the serializer.

![Diagram of UART Channel Control Registers]

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field</th>
<th>ARM/DSP Access</th>
<th>PRU Access</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15 - 12</td>
<td>Reserved</td>
<td>X</td>
<td>X</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>11 - 8</td>
<td>Serializer</td>
<td>W</td>
<td>R</td>
<td>0 - 15</td>
<td>The value indicates which serializer is mapped to this channel.</td>
</tr>
<tr>
<td>7 - 5</td>
<td>Reserved</td>
<td>X</td>
<td>X</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>4 - 3</td>
<td>ASPInstance</td>
<td>W</td>
<td>R</td>
<td>0 - 3</td>
<td>McAASP Instance. It takes the value 0 – 3.</td>
</tr>
<tr>
<td>2</td>
<td>serviceRequest</td>
<td>W</td>
<td>R/W</td>
<td>0</td>
<td>No Transmit or Receive Request. This is Reset by PRU.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>Transmit or Receive Request. This is set by ARM/DSP</td>
</tr>
<tr>
<td>1 - 0</td>
<td>Mode</td>
<td>W</td>
<td>R</td>
<td>00</td>
<td>Not Active</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>01</td>
<td>Transmit only</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10</td>
<td>Receive only</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>11</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
5.1.2 Configuration Register 1 (CHn_Config1)

This register holds the information of baud rate pre-scalar value for each channel.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field</th>
<th>ARM/DSP Access</th>
<th>PRU Access</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>OvrunIntrMask</td>
<td>W</td>
<td>R</td>
<td>0</td>
<td>Overrun Error Interrupt Masked</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>Overrun Error Interrupt Unmasked only if ErrIntr bit of UART Mask Register is set.</td>
</tr>
<tr>
<td>14</td>
<td>TimeoutIntrMask</td>
<td>W</td>
<td>R</td>
<td>0</td>
<td>Timeout Error Interrupt Masked</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>Timeout Interrupt Unmasked only if “ErrIntr” bit of UART Mask Register is set.</td>
</tr>
<tr>
<td>13</td>
<td>BIIIntrMask</td>
<td>W</td>
<td>R</td>
<td>0</td>
<td>Break Indicator Interrupt Masked</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>Break Indicator Interrupt Unmasked only if “ErrIntr” bit of UART Mask Register is set.</td>
</tr>
<tr>
<td>12</td>
<td>FEIntrMask</td>
<td>W</td>
<td>R</td>
<td>0</td>
<td>Framing Error Interrupt Masked</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>Framing Error Interrupt Unmasked only if ErrIntr bit of UART Mask Register is set.</td>
</tr>
<tr>
<td>11-10</td>
<td>Oversampling</td>
<td>W</td>
<td>R</td>
<td>00</td>
<td>No Oversampling</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>01</td>
<td>8 bits Oversampling</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10</td>
<td>16 bits Oversampling</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>11</td>
<td>Reserved</td>
</tr>
<tr>
<td>9-0</td>
<td>PreScalerValue</td>
<td>W</td>
<td>R</td>
<td>0-9</td>
<td>Maximum baud rate / $2^n$. Where $n$ is the “PreScalerValue”</td>
</tr>
</tbody>
</table>
### 5.1.3 Configuration Register 2 (CHn_Config2)

![Configuration Register 2 Diagram]

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field</th>
<th>ARM Access</th>
<th>PRU Access</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15 – 12</td>
<td>scratch_Parity</td>
<td>W</td>
<td>W/R</td>
<td>0-12</td>
<td>Temporary register for adding rx parity bits</td>
</tr>
<tr>
<td>11 – 8</td>
<td>DataLen</td>
<td>W</td>
<td>W/R</td>
<td>0 - 15</td>
<td>Length of transmit/receive buffer to be transmitted/received.</td>
</tr>
<tr>
<td>7 – 6</td>
<td>Parity</td>
<td>W</td>
<td>R</td>
<td>0 - 3</td>
<td>Parity supported Value 0 is None, Value 1 is Odd parity, Value 2 is Even parity</td>
</tr>
<tr>
<td>5 – 4</td>
<td>Reserved</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>Reserved</td>
</tr>
<tr>
<td>3 – 0</td>
<td>bitsPerChar</td>
<td>W</td>
<td>R</td>
<td>6 - 12</td>
<td>Number of bits per character</td>
</tr>
</tbody>
</table>
5.1.2 Transmit/Receive Registers
There are 8 channels per PRU and each channel will have transmit or receive registers state information. Each channel contains the control register, Status Register and transmit/receive buffer as shown below:

Soft UART Channel[x] TX/RX Registers

For each Soft UART channel has 1 byte of control register, 1 byte of status register and 4 bytes of transmit/receive data register is allocated.

5.2.2 TX/RX Status Register (CHn_TXRXStatus)

The register holds the information about transmit or receive status. These are the events at McASP side and updated by the PRU for specific serializer. The following layout shows the Status register bit descriptions:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field</th>
<th>ARM Access</th>
<th>PRU Access</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>Channel State</td>
<td>X</td>
<td>R/W</td>
<td>0</td>
<td>Channel (serializer) Inactive</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>Channel (serializer) Active</td>
</tr>
<tr>
<td>14</td>
<td>Reserved</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>Reserved</td>
</tr>
<tr>
<td>#</td>
<td>Register Name</td>
<td>Mode</td>
<td>Value</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>-------------------</td>
<td>------</td>
<td>-------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>PE MASK</td>
<td>R</td>
<td>W</td>
<td>0</td>
<td>No Parity error</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>Parity Error (RX)</td>
</tr>
<tr>
<td>7</td>
<td>PE</td>
<td>R</td>
<td>W</td>
<td>0</td>
<td>No Parity error</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>Parity Error (RX)</td>
</tr>
<tr>
<td>6</td>
<td>TO</td>
<td>R</td>
<td>W</td>
<td>0</td>
<td>Time-out</td>
</tr>
<tr>
<td>5</td>
<td>BI</td>
<td>R</td>
<td>W</td>
<td>0</td>
<td>No Break Indicator Error</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>Break Indicator Error (RX)</td>
</tr>
<tr>
<td>4</td>
<td>FE</td>
<td>R</td>
<td>W</td>
<td>0</td>
<td>No Framing error</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>Framing Error (RX)</td>
</tr>
<tr>
<td>3</td>
<td>TXRXRUNERR</td>
<td>R</td>
<td>W</td>
<td>0</td>
<td>No Error</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>If this bit is 0 and TXRXError bit is set then it is</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Unknown/other TX/RX errors</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Overrun in case of RX</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Underrun in case of TX</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>If there is any under/over run error, this bit is set along</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>with TXRXError.</td>
</tr>
<tr>
<td>2</td>
<td>TXRXError</td>
<td>R</td>
<td>W</td>
<td>0</td>
<td>Transmit/Receive no-error/Reset Value</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>Transmit/Receive Error</td>
</tr>
<tr>
<td>1</td>
<td>TXComplete/RXFIFOIndex</td>
<td>R</td>
<td>W</td>
<td>0</td>
<td>Transmit: Reset Value</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>Transmit Complete.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>This will be set by PRU on Transmit complete</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>Receive:</strong> Data Lower upper of FIFO.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Data in Upper Half of the FIFO</td>
</tr>
<tr>
<td></td>
<td>TXRXReady</td>
<td>R</td>
<td>R/W</td>
<td>0</td>
<td>Transmit:</td>
</tr>
<tr>
<td>----</td>
<td>-----------</td>
<td>-------</td>
<td>-----</td>
<td>----</td>
<td>-----------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Ready to Transmit. Reset by PRU</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TX Busy. This is set by PRU. Before interrupting for next transmission ARM has to check this bit.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R</td>
<td>R/W</td>
<td>0</td>
<td>Receive:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Ready for receive.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ARM/DSP reads the data and reset the bit to 0 informing PRU to receive next set of data.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Receive Busy. PRU will update this bit saying that the data is ready in the receive buffer.</td>
</tr>
</tbody>
</table>

### 5.2.2 Transmit/Receive Data Register (CHn_TXRXData)

This register holds the pointer to data buffer and the data buffer should be at ARM/DSP (outside of PRU DATA RAM) global address space. The data length will be configured in CHn_Config2 register and maximum allowed data length is 16 words.
5.2 UART Global Registers

5.2.1 UART Interrupt Mask Register

The ARM can configure this register to enable or disable any interrupts. Each channel will have interrupt mask bit.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field</th>
<th>ARM Access</th>
<th>PRU Access</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15-11</td>
<td>Reserved</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>Reserved</td>
</tr>
<tr>
<td>10</td>
<td>ConfIntr</td>
<td>W</td>
<td>R</td>
<td>0</td>
<td>Configuration Interrupt Mask bit</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Disable Configuration Interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>Enable Configuration Interrupt</td>
</tr>
<tr>
<td>9</td>
<td>ErrIntr</td>
<td>W</td>
<td>R</td>
<td>0</td>
<td>Error Interrupt Mask bit</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Disable Error Interrupts</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>Enable Error Interrupts</td>
</tr>
<tr>
<td>8</td>
<td>SoftwareIntr</td>
<td>W</td>
<td>R</td>
<td>0</td>
<td>Software Interrupt Mask bit</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Disable Software Interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>Enable Software Interrupt</td>
</tr>
<tr>
<td>7-0</td>
<td>ChannelIntr</td>
<td>W</td>
<td>R</td>
<td>0-7</td>
<td>Channel Interrupt Mask bit</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Value 0 for specific channel is to disable interrupt from PRU SUART channel</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>to ARM/DSP</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Value 1 for specific channel is to enable interrupts from PRU SUART channel</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>to ARM/DSP</td>
</tr>
</tbody>
</table>
5.2.1 UART Interrupt Status Register

ARM request the PRU to act based on the control register. The PRU service the ARM request and populates the Status registers and appropriate interrupt status bit is set and then the PRU Interrupts the ARM to take further action based on the status.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field</th>
<th>ARM Access</th>
<th>PRU Access</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15-11</td>
<td>Reserved</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>Reserved</td>
</tr>
<tr>
<td>10</td>
<td>ConfIntr</td>
<td>R</td>
<td>W</td>
<td>0</td>
<td>Configuration Interrupt Status bit</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Reset state</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>Configuration complete</td>
</tr>
<tr>
<td>9</td>
<td>ErrIntr</td>
<td>R</td>
<td>W</td>
<td>0</td>
<td>Error Interrupt Status bit</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Reset state</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>This bit is set if there is any error in soft uart channels</td>
</tr>
<tr>
<td>8</td>
<td>SoftwareIntr</td>
<td>R</td>
<td>W</td>
<td>0</td>
<td>Software Interrupt Status bit</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Reset state</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>Software Reset Interrupt serviced</td>
</tr>
<tr>
<td>7 - 0</td>
<td>ChannelIntr</td>
<td>R</td>
<td>W</td>
<td>0 - 7</td>
<td>Channel Interrupt Status bit</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Value 0 is Reset state</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Value 1 for specific channel is serviced the interrupt</td>
</tr>
</tbody>
</table>
The following section provides the Communication interface between ARM to PRU. The data and status availability on the PRU side would be intimated to the core through interrupts. The below diagram depicts the layering and data flow between PRU and Core.

The status registers would be allocated a specific location in Data RAM of the PRU and based on the API call the data would be presented to the caller.

Data Structures

6.1.1.

6.1.2. suart_config
Structure used to initialize the UART. This structure is defined in the file suart_api.h.

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uint8 TXSerializer</td>
<td>It takes the value of the serializer. It takes the value 0-3 range.</td>
</tr>
<tr>
<td>Uint8 RXSerializer</td>
<td>It takes the value of the serializer. It takes the value 0-3 range.</td>
</tr>
<tr>
<td>Uint16 txClkDivisor</td>
<td>Divisor (CLKXDIV* HCLKXDIV) value to generate the appropriate baud rate.</td>
</tr>
<tr>
<td></td>
<td>Note: Baud Rate (Range: 300 to 230400).</td>
</tr>
</tbody>
</table>
### UART_Config

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uint16 txClkDivisor</td>
<td>Divisor (CLKXDIV* HCLKXDIV) value to generate the appropriate baud rate</td>
</tr>
<tr>
<td>Uint16 txBitsPerChar</td>
<td>Bits per Character (Range: 6 to 12)</td>
</tr>
<tr>
<td></td>
<td>Valid symbolic values are:</td>
</tr>
<tr>
<td></td>
<td>SUART_DATA_BITS6 6 bits per character</td>
</tr>
<tr>
<td></td>
<td>SUART_DATA_BITS7 7 bits per character</td>
</tr>
<tr>
<td></td>
<td>SUART_DATA_BITS8 8 bits per character</td>
</tr>
<tr>
<td></td>
<td>SUART_DATA_BITS9 9 bits per character</td>
</tr>
<tr>
<td></td>
<td>SUART_DATA_BITS10 10 bits per character</td>
</tr>
<tr>
<td></td>
<td>SUART_DATA_BITS11 11 bits per character</td>
</tr>
<tr>
<td></td>
<td>SUART_DATA_BITS12 12 bits per character</td>
</tr>
<tr>
<td>Uint16 rxBitsPerChar</td>
<td>Bits per Character (Range: 6 to 12)</td>
</tr>
<tr>
<td></td>
<td>Valid symbolic values are:</td>
</tr>
<tr>
<td></td>
<td>SUART_DATA_BITS6 6 bits per character</td>
</tr>
<tr>
<td></td>
<td>SUART_DATA_BITS7 7 bits per character</td>
</tr>
<tr>
<td></td>
<td>SUART_DATA_BITS8 8 bits per character</td>
</tr>
<tr>
<td></td>
<td>SUART_DATA_BITS9 9 bits per character</td>
</tr>
<tr>
<td></td>
<td>SUART_DATA_BITS10 10 bits per character</td>
</tr>
<tr>
<td></td>
<td>SUART_DATA_BITS11 11 bits per character</td>
</tr>
<tr>
<td></td>
<td>SUART_DATA_BITS12 12 bits per character</td>
</tr>
<tr>
<td>Oversampling</td>
<td>Oversampling rate</td>
</tr>
<tr>
<td></td>
<td>SUART_8X_OVRSMPLE</td>
</tr>
<tr>
<td></td>
<td>SUART_16X_OVRSMPLE</td>
</tr>
<tr>
<td>BIIntrMask</td>
<td>Break Indicator Interrupt Mask Bit</td>
</tr>
<tr>
<td>FEIntrMask</td>
<td>Framing Error Interrupt Mask Bit</td>
</tr>
<tr>
<td>PEIntrMask</td>
<td>Parity Error Interrupt Mask Bit</td>
</tr>
</tbody>
</table>

UART_Config structure used to initialize the specified UART. After created and initialized the structure, it is passed to the PRU_SoftUart_SetConfig() function to configure the required UART.

### 6.1.2. suart_struct_handle

Structure used for UART Handle. This structure is defined in the file suart_api.h.

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uint16 uartNum</td>
<td>UART number (Range 1 to 16)</td>
</tr>
<tr>
<td></td>
<td>Valid Values are</td>
</tr>
<tr>
<td></td>
<td>SOFT_UART_NUM_n</td>
</tr>
<tr>
<td></td>
<td>Where n is 1 to 4</td>
</tr>
<tr>
<td>Uint16 uartType</td>
<td>Type of the UART,</td>
</tr>
<tr>
<td></td>
<td>• PRU_SUART_TX_RX</td>
</tr>
<tr>
<td>Uint16 uartTxChannel</td>
<td>Soft UART Channel for Transmission</td>
</tr>
<tr>
<td>Uint16 uartRxChannel</td>
<td>Soft UART Channel for Reception</td>
</tr>
<tr>
<td>Uint16 uartStatus</td>
<td>Status of the UART</td>
</tr>
</tbody>
</table>
**6.1.3. tSuartInfo**

Structure used for storing configurable parameters for each SUART. This is defined in `am335x_suart_board.h`.

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uint32 suartNum</td>
<td>UART number (Range 1 to NR_SUART+1)</td>
</tr>
<tr>
<td>Uint32 pruNum</td>
<td>PRU on which the SUART is running PRUSS_NUM0 PRUSS_NUM1</td>
</tr>
<tr>
<td>Uint32 txPruChn</td>
<td>PRU channel for Transmit (0 to 3)</td>
</tr>
<tr>
<td>Uint32 txPruChn</td>
<td>PRU channel for Receive (0 to 3)</td>
</tr>
<tr>
<td>Uint32 txSysEvent</td>
<td>System Event that is used to inform Transmit</td>
</tr>
<tr>
<td>Uint32 rxSysEvent</td>
<td>System Event that is used to inform Receive</td>
</tr>
<tr>
<td>Uint32 ctxBase</td>
<td>Soft UART context base</td>
</tr>
<tr>
<td>Uint32 fmtDataBase</td>
<td>Formatted data base address</td>
</tr>
<tr>
<td>Uint32 mcaspNum</td>
<td>McASP number where SUART is running</td>
</tr>
<tr>
<td>Uint32 txSerializer</td>
<td>Soft Uart Serializer for Transmit PRU_SUART_SERIALIZER_n (n from 0 to 3)</td>
</tr>
<tr>
<td>Uint32 rxSerializer</td>
<td>Soft Uart Serializer for Receive PRU_SUART_SERIALIZER_n (n from 0 to 3)</td>
</tr>
</tbody>
</table>

**6.2. SUART Configurable Parameters**

**6.2.1. Number of SUARTS**

The number of SUARTs is configurable in `am335x_suart_board.h` header file in the SUART driver source code. Edit the macro `NR_SUART` in the `drivers/tty/serial/am335x_pru_suart/am335x_suart_board.h`. Value of `NR_SUART` should be an integer ranging from 1 to 4.

**6.2.2. Enabling PRUs**

Each PRU will support 2 SUARTS. Based on the number of SUARTs required and hardware design enable the PRUs. PRU is configurable in the `am335x_suart_board.h` header file. Edit the macros **PRU_MODE**, **PRU1_MODE** in `drivers/tty/serial/am335x_pru_suart/am335x_suart_board.h` to enable/disable usage of a PRU. Value of these macros should be **PRU_MODE_RX_TX_BOTH** if the PRU need to be enabled, else it should be **PRU_MODE_INVALID** to disable the PRU.

**6.2.3. Enabling the McASPs**

McASPs need to be enabled through menuconfig options. Enable the SUART Driver

Device Drivers ---> Character devices ---> Serial drivers ---> PRU based SUART emulation

Then there will be options to choose the McASPs, Based on the number of SUARTs enable the McASPs.
6.2.4. **PRU to McASP mapping**
By default in the driver McASP1 is mapped to PRU0 and McASP0 is mapped to PRU1. The mapping of McASP to PRU can be changed by modifying the `PRU_INTC_CHANMAP8_FULL` and `PRU_INTC_CHANMAP13_FULL` defines in `include/linux/mfd/pruss.h`

For McASP1 to PRU0, McASP0 to PRU1 mapping

```c
#define PRU_INTC_CHANMAP8_FULL (0x00000000) // 35 34 33 32 - McASP1 to PRU0
#define PRU_INTC_CHANMAP8_FULL (0x00010100) // 35 34 33 32 - McASP1 to PRU1
#define PRU_INTC_CHANMAP13_FULL (0x01010000) // 55 54 53 52 - McASP0 to PRU1
#define PRU_INTC_CHANMAP13_FULL (0x00000000) // 55 54 53 52 - McASP0 to PRU0
```

For McASP1 to PRU0, McASP0 to PRU1 mapping

```c
#define PRU_INTC_CHANMAP8_FULL (0x00000000) // 35 34 33 32 - McASP1 to PRU0
#define PRU_INTC_CHANMAP8_FULL (0x00010100) // 35 34 33 32 - McASP1 to PRU1
#define PRU_INTC_CHANMAP13_FULL (0x01010000) // 55 54 53 52 - McASP0 to PRU1
#define PRU_INTC_CHANMAP13_FULL (0x00000000) // 55 54 53 52 - McASP0 to PRU0
```

When McASP to PRU mapping is changed, make sure to change the gSuartInfo structure with proper McASP and PRU number for all the SUARTS.

6.2.5. **Assigning McASP serializers to SUART**
Serializers for a specific SUART should be specified gSuartInfo structure fields txSerializer and rxSerializer. The serializer numbers specified should belong to the McASP instance assigned to the PRU core on which SUART is running. Value for the txSerializer or rxSerializer in a McASP will be `PRU_SUART_SERIALIZER_x` (where x can be any number from 0 to 3. If McASP serializer is not connected to a specific SUART in hardware, then the values of serializer should be set to `PRU_SUART_SERIALIZER_NONE`.

### 6.2Function

#### 6.2.1 pru_softuart_init

```c
short pru_softuart_init (struct device *dev, 
unsigned int txBaudValue, 
unsigned int rxBaudValue, 
unsigned int oversampling, 
unsigned char *pru_suart_emu_code, 
unsigned int fw_size, 
arm_pru_iomap * arm_iomap_pru)
```

**Arguments**
- `dev` - Device node
- `txBaudValue` - Default Base Baud Rate
- `rxBaudValue` - Default Base Baud Rate
- `oversampling` - RX oversampling Factor
- `pru_suart_emu_code` - PRU Firmware code address
- `fw_size` - PRU Firmware Size
- `arm_iomap_pru` - IO Base Address for PRU, MCASP, TX & RX Buffer

**Return Value**
- `PRU_SUART_SUCCESS` - returns zero on Success.
- Non-zero on Failure, return values are
  - `PRU_SUART_FAILURE`
Pre Condition
None.

Description
This function initializes the PRU and loads the Soft UART to PRU. This should be called only once before starting the Soft UART activities.

6.2.2 pru_set_fifo_timeout

void pru_set_fifo_timeout (struct device *dev, Uint32 timeout)

Arguments
- dev - Device node
- timeout - Wait for Timeout (in ms) after receiving a character.

Return Value
- None

Pre Condition
None.

Description
This function sets the timeout (in ms), pru waits for timeout duration once receiving a character, before it raises timeout interrupt.

6.2.3 pru_mcasp_deinit

void pru_mcasp_deinit (void)

Arguments
- None.

Return Value
- None

Pre Condition
pru_softuart_init() function must be called before calling this function.

Description
This function resets the MCASP before MCASP clock is disabled.

6.2.4 pru_softuart_deinit

short pru_softuart_deinit(struct device *dev)

Arguments
- dev - Device node
Return Value
- PRU_SOFTWARE_SUCCESS - returns zero on success
- Non-zero on Failure, return values are
  - -1

Pre Condition
pru_softuart_init() function must be called before calling this function.

Description
This function reset and halts the PRU before PRU clock is disabled.

6.2.5 PRU_SoftUart_Open

short pru_softuart_open (suart_handle hSuart)

Arguments
- hSuart - handle to UART.

Return Value
- PRU_SOFTWARE_SUCCESS - returns zero on success
- Non-zero on Failure, return values are
  - SUART_INVALID_UART_NUM

Pre Condition
None.

Description
This function opens and initializes the state of the UART and other parameter of UART handle.

Note: The Soft UART Channels are mapped to serializers in the board specific file am335x_suart_board.h.

6.2.6 pru_softuart_close

short pru_softuart_close (suart_handle hUart)

Arguments
- hUart - handle to UART.

Return Value
- returns PRU_SOFTWARE_SUCCESS on successful handle close
- Returns non-zero value on failure, return values are
  - PRU_SOFTWARE_ERR_HANDLE_INVALID

Pre Condition
PRU_SoftUart_Open() function must be called before calling this function.

Description
This function changes the state of Soft UART and invalidates the entries in UART handle.
6.2.7 PRU_SoftUart_SetBaud

short pru_softuart_setbaud (struct device *dev,
                         suart_handle hUart,
                         unsigned short txClkDivisor,
                         unsigned short rxClkDivisor)

Arguments
- Dev - Device node
- hUart - handle to UART.
- txClkDivisor - TX Baud Prescaler value
- rxClkDivisor - TR Baud Prescaler value

Return Value
- PRU_SOFTUART_SUCCESS - returns zero on success
- Returns non-zero value on failure, return values are
  o PRU_SOFTUART_ERR_DEVICE_NOT_OPEN
  o PRU_SOFTUART_ERR_PARAMETER_INVALID
  o PRU_MODE_INVALID

Pre Condition
PRU_SoftUart_Open() function must be called before calling this function.

Description
This function is to set the baud rate for specified SOFT UART.

6.2.8 PRU_SoftUart_SetParity

short pru_softuart_setparity (suart_handle hUart,
                              unsigned short parity);

Arguments
- hUart - handle to UART.
- Parity - Parity value

Return Value
- PRU_SOFTUART_SUCCESS - returns zero on success
- Returns non-zero value on failure, return values are
  o PRU_SOFTUART_ERR_DEVICE_NOT_OPEN
  o PRU_SOFTUART_ERR_PARAMETER_INVALID
  o PRU_MODE_INVALID

Pre Condition
PRU_SoftUart_Open() function must be called before calling this function.

Description
This function is to set the parity for specified SOFT UART.
6.2.9 PRU_SoftUart_SetDataBits

```
short pru_softuart_setdatabits (struct device *dev,
    suart_handle hUart,
    unsigned short txDataBits,
    unsigned short rxDataBits)
```

**Arguments**
- Dev - Device node
- hUart - handle to UART.
- txDataBits - Number of bits per character should be configurable from 6 to 12 bit i.e. (6, 7, 8, 9, 10, 11, and 12)
- rxDataBits - Number of bits per character should be configurable from 6 to 12 bit i.e. (6, 7, 8, 9, 10, 11, and 12)

**Return Value**
- PRU_SOF TUART_SUCCESS - returns zero on success
- Returns non-zero value on failure, return values are
  - PRU_SOF TUART_ERR_DEVICE_NOT_OPEN
  - PRU_SOF TUART_ERR_PARAMETER_INVALID
  - PRU_MODE_INVALID

**Pre Condition**
PRU_SoftUart_Open() function must be called before calling this function.

**Description**
This function sets number bits per character for Soft UART.

6.2.10 pru_softuart_setconfig

```
short pru_softuart_setconfig (struct device *dev,
    suart_handle hUart,
    suart_config *configUart)
```

**Arguments**
- dev - Device node
- hUart - handle to UART.
- configUart - point to UART_Config structure to configure UART

**Return Value**
- SUART_SUCCESS - returns zero on success
- Returns non-zero value on failure, return values are
  - PRU_SUART_ERR_HANDLE_INVALID
  - PRU_MODE_INVALID

**Pre Condition**
PRU_SoftUart_Open() function must be called before calling this function.

**Description**
This function configures the baud rate, bits per character, stop bits and parity bit for specified UART.

6.2.11 pru_softuart_getTxDataLen

short pru_softuart_getTxDataLen (struct device *dev, suart_handle hUart)

Arguments
- dev - Device node
- hUart - handle to UART.

Return Value
- DataLength - returns actual data length on success.
- Returns non-zero value on failure, return values are
  - PRU_SUART_ERR_HANDLE_INVALID
  - PRU_MODE_INVALID

Pre Condition
PRU_SoftUart_Open() function must be called before calling this function.

Description
This function uses UART handle passed to it and reads the actual DataLength (number of characters to transmit) of the TX channel of the Soft UART before the TX channel issues the interrupt to ARM/DSP for next request of transmission.

6.2.12 pru_softuart_getRxDataLen

short pru_softuart_getRxDataLen (struct device *dev, suart_handle hUart)

Arguments
- dev - Device node
- hUart - handle to UART.

Return Value
- DataLength - returns actual data length on success.
- Returns non-zero value on failure, return values are
  - PRU_SUART_ERR_HANDLE_INVALID
  - PRU_MODE_INVALID

Pre Condition
PRU_SoftUart_Open() function must be called before calling this function.

Description
This function uses UART handle passed to it and reads the actual DataLength (number of characters to receive) of the RX channel of the Soft UART before the RX channel issues the interrupt to ARM/DSP to read the received data.
6.2.13 PRU_SoftUart_GetConfig

short pru_softuart_getconfig (struct device *dev,
               suart_handle hUart,
               suart_config * configUart)

Arguments
- Dev - Device node
- hUart - handle to UART.
- configUart – point to UART_Config structure

Return Value
- SUART_SUCCESS - returns zero on success
- Returns non-zero value on failure, return values are
  o PRU_SUART_ERR_HANDLE_INVALID
  o PRU_MODE_INVALID

Pre Condition
PRU_SoftUart_Open() function must be called before calling this function.

Description
This function is to get the current configuration for the SOFT UART used by handle. The function reads the actual PRU channel registers and fields and storing them in the UART_Config structure.

6.2.14 pru_softuart_write

short pru_softuart_write (struct device *dev,
               suart_handle hUart,
               unsigned int *ptTxDataBuf,
               unsigned short dataLen)

Arguments
- dev - Device node
- hUart - handle to UART.
- ptTxDataBuf - Pointer to the data buffer containing the data to be written
- dataLen - Length of data in the buffer to be written

Return Value
- SUART_SUCCESS - returns zero on success
- Returns non-zero value on failure, return values are
  o PRU_SUART_ERR_HANDLE_INVALID
  o PRU_MODE_INVALID

Pre Condition
PRU_SoftUart_Open() function must be called before calling this function.

Description
This register holds the pointer to data buffer and the data buffer should be at ARM/DSP (outside of PRU DATA RAM) global address space. The data length will be configured in CHn_Config2 register and maximum allowed data length is 16 words.
6.2.15 pru_softuart_read

```c
short pru_softuart_read (struct device *dev,
    suart_handle hUart,
    unsigned int *ptDataBuf,
    unsigned short dataLen)
```

**Arguments**
- **dev** - Device node
- **hUart** - handle to UART.
- **ptDataBuf** - Pointer to the data buffer in which the received data will be written.
- **dataLen** - Length of the data buffer in which the read data will be returned.

**Return Value**
- **SUART_SUCCESS** - returns zero on success
- Returns non-zero value on failure, return values are
  - **PRU_SUART_ERR_HANDLE_INVALID**
  - **PRU_MODE_INVALID**

**Pre Condition**
PRU_SoftUart_Open() function must be called before calling this function.

**Description**
This register holds the pointer to data buffer and the data buffer should be at ARM/DSP (outside of PRU DATA RAM) global address space. The data length will be configured in CHn_Config2 register and maximum allowed data length is 16 words.

6.2.16 pru_softuart_read_data

```c
short pru_softuart_read_data (struct device *dev,
    suart_handle hUart
    Uint8 * pDataBuffer,
    Int32 s32MaxLen,
    Uint32 * pu32DataRead)
```

**Arguments**
- **dev** - Device node
- **hUart** - handle to UART.
- **pDataBuffer** - Pointer to the data buffer in which the received data has been written.
- **s32MaxLen** - Maximum Data that can be read from pDataBuffer and copied to pu32DataRead
- **pu32DataRead** - Pointer to the data buffer in which where received data is to copied.

**Return Value**
- **PRU_SUART_SUCCESS** - returns zero on success
- Returns non-zero value on failure, return values are
  - **PRU_SUART_ERR_HANDLE_INVALID**
  - **PRU_MODE_INVALID**

**Pre Condition**
PRU_SoftUart_Open() function must be called before calling this function.

**Description**
This function reads the data received by the RX channel of Soft UART and copies it to the ARM/DSP local memory.

### 6.2.17 pru_softuart_getTxStatus

**short pru_softuart_getTxStatus (struct device *dev, suart_handle hUart)**

**Arguments**
- dev - Device node
- hUart - handle to UART.

**Return Value**
- Returns TXRXSTATUS register on success.
- Returns non-zero value on failure, return values are
  - PRU_SUART_ERR_HANDLE_INVALID
  - PRU_MODE_INVALID

**Pre Condition**
PRU_SoftUart_Open() function must be called before calling this function.

**Description**
This function reads TXRXSTATUS register of the Soft UART's TX channel referred by the UART handle and return the same.
6.2.18 pru_softuart_clrTxStatus

short pru_softuart_clrTxStatus (struct device *dev, suart_handle hUart)

Arguments
- Dev - Device node
- hUart - handle to UART.

Return Value
- SUART_SUCCESS – returns zero on success.
- Returns non-zero value on failure, return values are
  - PRU_SUART_ERR_HANDLE_INVALID
  - PRU_MODE_INVALID

Pre Condition
PRU_SoftUart_Open() function must be called before calling this function.

Description
This function clears the PRU_SOFTUART_RX_COMPLETE bit in the TXRXSTATUS register of the Soft UART’s TX channel referred by the UART handle one the TX complete interrupt has occurred.

6.2.19 pru_softuart_getRxStatus

short pru_softuart_getRxStatus (struct device *dev, suart_handle hUart)

Arguments
- dev - Device node
- hUart - handle to UART.

Return Value
- Returns TXRXSTATUS register on success.
- Returns non-zero value on failure, return values are
  - PRU_SUART_ERR_HANDLE_INVALID
  - PRU_MODE_INVALID

Pre Condition
PRU_SoftUart_Open() function must be called before calling this function.

Description
This function reads TXRXSTATUS register of the Soft UART’s TX channel referred by the UART handle and return the same.

6.2.20 pru_softuart_clrRxFifo

short pru_softuart_clrRxFifo (struct device *dev, suart_handle hUart)

Arguments
- dev - Device node
• **hUart** - handle to UART.

**Return Value**
- Returns TXRXSTATUS register on success.
- Returns non-zero value on failure, return values are
  - PRU_SUART_ERR_HANDLE_INVALID
  - PRU_MODE_INVALID

**Pre Condition**
PRU_SoftUart_Open() function must be called before calling this function.

**Description**
This function is called in case of timeout interrupt, it resets the Soft UART Rx channel BYTESDONECNTR so that FIFO is reset, and issues a fresh service request for receive to the PRU on this Soft UART.

### 6.2.21 pru_softuart_clrRxStatus

**short pru_softuart_clrRxStatus (struct device *dev, suart_handle hUart)**

**Arguments**
- dev - Device node
- hUart - handle to UART.

**Return Value**
- SUART_SUCCESS – returns zero on success.
- Returns non-zero value on failure, return values are
  - PRU_SUART_ERR_HANDLE_INVALID
  - PRU_MODE_INVALID

**Pre Condition**
PRU_SoftUart_Open() function must be called before calling this function.

**Description**
This function clears the all the bits in TXRXSTATUS register of RX channel of Soft UART except for the service request and FIFO index bit.

### 6.2.22 pru_softuart_get_isrstatus

**short pru_softuart_get_isrstatus (struct device *dev, , unsigned short uartNum, unsigned short *txrxFlag)**

**Arguments**
- dev - Device node
- uartNum - Soft UART number which interrupt status requested (Soft UART number can be from 1 to 8.
- txrxFlag - Pointer to buffer to pass TX/RX interrupt status for Soft UART.

**Return Value**
- SUART_SUCCESS – returns zero on success.
• Returns non-zero value on failure

Pre Condition
pru_softuart_init() function must be called before calling this function.

Description
This function gets the Global Interrupt status register in PRU interrupt controller check the TX/Rx interrupt for the specified Soft UART and if interrupt occurred, acknowledges the interrupt and update the txrxFlag.

6.2.23 pru_intr_clr_isrstatus

int pru_intr_clr_isrstatus (struct device *dev,
unsigned short uartNum,
unsigned int txrxmode)

Arguments
• dev - Device node
• uartNum - UART number which TX/RX status bit is to clear.
• txrxmode – TX or RX mode.

Return Value
• SUART_SUCCESS – returns zero on success.
• Returns non-zero value on failure, return values are
  o PRU_MODE_INVALID

Pre Condition
PRU_SoftUart_Open() function must be called before calling this function.

Description
This function clears the TX/ RX bit in ISR status register corresponding to Soft UART depending upon the mode passed TX or RX.

6.2.24 suart_pru_to_host_intr_enable

int suart_pru_to_host_intr_enable (struct device *dev,
unsigned short uartNum,
unsigned int txrxmode,
int s32Flag)

Arguments
• dev - Device node
• uartNum - Uart Number which RX/TX interrupt to enabled / Disabled
• txrxmode – TX or RX interrupt
• s32Flag - Interrupt to be disabled or enabled.

Return Value
• SUART_SUCCESS – returns zero on success.
• Returns non-zero value on failure

Pre Condition
pru_softuart_init() function must be called before calling this function.

**Description**
This function enables the TX/ RX interrupt in the PRU Interrupt Controller depending on the mode and flag passed for the soft UART.

### 6.2.25 suart_intr_setmask

```c
int suart_intr_setmask (struct device *dev, 
unsigned short uartNum, 
unsigned int txrxmode, 
unsigned int intrmask)
```

**Arguments**
- `dev` - Device node
- `uartNum` - Uart Number which RX/TX interrupt is to enabled in mask register from PRU to ARM.
- `txrxmode` - TX or RX interrupt to masked.
- `intrmask` - Complete / Error/ Timeout for Soft UART interrupt to be enabled.

**Return Value**
- `SUART_SUCCESS` – returns zero on success.
- Returns non-zero value on failure,
  - `SUART_INVALID_UART_NUM`
  - `PRU_MODE_INVALID`

**Pre Condition**
pru_softuart_init() function must be called before calling this function.

**Description**
This function enables the Complete / Error / Timeout interrupt from PRU to ARM/DSP for RX/TX depending on txrxmode for the specified UART.

### 6.2.26 suart_intr_clrmask

```c
int suart_intr_clrmask (struct device *dev, 
unsigned short uartNum, 
unsigned int txrxmode, 
unsigned int intrmask)
```

**Arguments**
- `dev` - Device node
- `uartNum` - Uart Number which RX/TX interrupt is to disabled in mask register from PRU to ARM.
- `txrxmode` - TX or RX interrupt to masked.
- `intrmask` - Complete / Error/ Timeout for Soft UART interrupt to be disabled.

**Return Value**
- `SUART_SUCCESS` – returns zero on success.
• Returns non-zero value on failure,
  o SUART_INVALID_UART_NUM
  o PRU_MODE_INVALID

Pre Condition
pru_softuart_init () function must be called before calling this function.

Description
This function disables the Complete / Error / Timeout interrupt from PRU to ARM/DSP for RX/TX depending on txrxmode for the specified UART.
Firmware Flow Diagrams

Figure A-1: Main loop
Figure A-4: TxServiceReqHndlr flow-chart

Figure A-5: Flow chart for RxServiceReqHndlr
Figure A-6: Flow chart for RxIntrHndlr 1
Figure A-7: Flow chart for RxIntrHndlr continued
Figure A-8: Flow Chart for TxIntrHndlr
Figure A-9: Flow chart for TxIntrHndlr Continued
Figure A-10: Flow Chart for TxIntrHndlr continued