Using the FPSLIC™ UARTs in C

Features

- **Setup and Use of the FPSLIC UARTs**
- **Code Examples for Polled and Interrupt Controlled UARTs**
- **Compact Code**
- **C Code Included for AT94K**

Introduction

This application note describes how to set up and use the UARTs present in most FPSLIC devices. C code examples are included for polled and interrupt controlled UART applications.

Polled UARTs

The application is continuously checking the UDREn bit in the UARTn Control and Status Register A (UCSRnA) to control when the UART has finished sending a byte. When receiving data, the application is continuously checking the RXCn bit in the UARTn Control and Status Register A to control when the UART has completed receiving a byte.

Interrupt Controlled UARTs

The UARTn generates an interrupt when the UARTn has finished transmitting or receiving a byte. The interrupt handling routine uses modulo 2n addressing of circular buffers for buffering incoming and outgoing data. The buffer sizes must be defined before using the routines. Set the UARTn_RX_BUFFER_SIZE and the UARTn_TX_BUFFER_SIZE variables to the buffer size in bytes. Note that these variables must be a power of two. If not, a compiler error message will be flagged.

An extra function is added to the UART2.C example code. The DataInReceiveBuffer returns a zero if the receive buffer does not contain any data. This function does, in contrast to the ReceiveByte function, not wait for incoming data, but returns immediately the status of the buffer.

Note: This routine does not return the number of bytes in the buffer.

Table 1. Polled vs. Interrupt Controlled UART Comparison

Polled UART	Interrupt Controlled UART
Compact Code	Reasonable Code Size
Application Busy while Communicating	Application Free while Communicating

FPSLIC™

Polled and Interrupt Controlled UARTs

Application Note

Rev. 1974A–06/01

Usage

Both examples use the same set of routines. If other devices than the AT94K are used, the included file in the code must be changed accordingly.

```
void initUARTn(unsigned char baudRate);
```

Enables UARTn and sets the baud rate. Using baud rates that differ more than ±0.5% is not recommended. Please refer to the UART section in the datasheet for selecting the baud rate. The value passed to this function will be written to the UARTn Baud Rate registers.

```
unsigned char rxByteUARTn(void);
```

Waits for one byte to be received and returns its value.

```
void txByteUARTn(unsigned char data);
```

Waits for transmission to be allowed, sends byte given as parameter to the UARTn transmitter and returns.

```
unsigned char dataInReceiveBuffer(void);
```

Returns zero (0) if the receive buffer is empty.

Example 1: UART.C (Polled)

```
#include <ioat94k.h>

/* Prototypes */
void initUART0(unsigned int baudRate);
unsigned char rxByteUART0(void);
void txByteUART0(unsigned char data);

/* Main - Simple Test Program */
void main(void)
  {
  initUART0(12);                                  /* Baud Rate to 19,200 bps using a 4 MHz Crystal */

  for(;;)                                         /* Forever */
  {
    txByteUART0(rxByteUART0());                   /* Echo the rx character */
  }
}

/* initUART0 */
void initUART0(unsigned int baudRate)
{
  UBRRHI |= (((baudRate) >> 8) & 0x000F);
  UBRR0  |= ((baudRate) & 0x00FF));
  UCSR0B |= ((1 << RXEN0) | (1 << TXEN0));
}

/* rxByteUART0 */
unsigned char rxByteUART0(void)
{
  while(!(UCSR0A & (1 << RXC0)));                 /* Wait: incoming data */
  return UDR1;                                    /* Return the data */
}

/* txByteUART0 */
void txByteUART0(unsigned char data)
{
  while(!(UCSR0A & (1 << UDRE1)));                /* Wait: empty tx buffer */
  UDR1 = data;                                    * Start tx operation */
}
```

Example 2: UART.C (Interrupt)

```
#include <ioat94k.h>


/*UART Buffer Defines */
#define UART0_RX_BUFFER_SIZE128
#define UART0_RX_BUFFER_MASK(UART0_RX_BUFFER_SIZE -1)
#if (UART0_RX_BUFFER_SIZE & UART0_RX_BUFFER_MASK)
#error RX Buffer Size NOT a Power of 2
#endif

#define UART0_TX_BUFFER_SIZE128
#define UART0_TX_BUFFER_MASK(UART0_TX_BUFFER_SIZE -1)
#if (UART0_TX_BUFFER_SIZE & UART0_TX_BUFFER_MASK)
#error TX Buffer Size NOT a Power of 2
#endif


/* Static Variables */
static unsigned char UART0_rxBuffer[UART0_RX_BUFFER_SIZE];
static volatile unsigned char UART0_rxHead;
static volatile unsigned char UART0_rxTail;

static volatile unsigned char UART0_txBuffer[UART_TX_BUFFER_SIZE];
static volatile unsigned char UART0_txHead;
static volatile unsigned char UART0_txTail;


/* Prototypes */
void initUART0(unsigned int baudRate);
unsigned char rxByteUART0(void);
void txByteUART0(unsigned char data);
unsigned char dataInRxBuffer(void);


/* Main - Simple Test Program */
void main(void)
{
  initUART0(12);                                  /* Baud Rate to 19,200 bps using a 4 MHz Crystal */

  SREG |= 0x80;                                   /* Enable Interrupts */

  for(;;)                                         /* Forever */
  {
    txByteUART0(rxByteUART0());                   /* Echo the rx character */
  }
}


/* initUART0 */
```

```
void initUART0(unsigned int baudRate)
{
  /* Set the Baud Rate */
  UBRRHI |= (((baudRate) >> 8) & 0x000F);
  UBRR0  |= ((baudRate) & 0x00FF));
  UCSR0B |= ((1 << RXEN0) | (1 << TXEN0) | (1 << RXCIE0));

  /* Flush the rx Buffer */
  UART0_rxTail = 0;
  UART0_rxHead = 0;
  UART0_txTail = 0;
  UART0_txHead = 0;
}

/* rxByteUART0 */
unsigned char rxByteUART0(void)
{
  unsigned char tmpTail;
  while(UART0_rxHead == UART0_rxTail);
  tmpTail = (UART0_rxTail + 1) & UART0_RX_BUFFER_MASK;
  UART0_rxTail = tmpTail;
  Return UART0_rxBuffer[tmpTail];
}

/* txByteUART0 */
void txByteUART0(unsigned char data)
{
  unsigned char tmpHead;
  tmpHead = (UART0_txHead + 1) & UART_TX_BUFFER_MASK;
  while(tmpHead == UART0_txTail);
  UART0_txBuffer[tmpHead] = data;
  UART0_txHead = tmpHead;
  UCSR0B |= (1 << UDRIE0);
}

/* dataInRxBuffer */
unsigned char dataInRx_buffer(void)
{
  return(UART0_rxHead != UART0_rxTail);
}


/* Interrupt Handlers */
#pragma vector = UART0_RXC
static __interrupt void UART0_RXC_interrupt(void)
{
  unsigned char data, tmpHead;

  data = UDR0;

  tmpHead = (UART0_rxHead + 1) & UART0_RX_BUFFER_MASK;
```

```
  UART0_rxHead = tmpHead;

  if(tmpHead == UART0_rxTail)
  {
    /* ERROR:  RX Buffer Overflow */
  }

  UART0_rxBuffer[tmpHead] = data;
}

#pragma vector = UART0_DRE
static __interrupt void UART0_DRE_interrupt(void)
{
  unsigned char tmpTail;

  if(UART0_txHead != UART0_txTail)
  {
    tmpTail = (UART0_txTail + 1) & UART_TX_BUFFER_MASK;
    UART_txTail = tmpTail;
    UDR0 = UART_txBuffer[tmpTail];
  }
  else
  {
    UCSR0B &= ~(1 << UDRIE0);
  }
}
```

Atmel Headquarters

Corporate Headquarters
2325 Orchard Parkway
San Jose, CA 95131
TEL (408) 441-0311
FAX (408) 487-2600

Europe
Atmel SarL
Route des Arsenaux 41
Casa Postale 80
CH-1705 Fribourg
Switzerland
TEL (41) 26-426-5555
FAX (41) 26-426-5500

Asia
Atmel Asia, Ltd.
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimhatsui
East Kowloon
Hong Kong
TEL (852) 2721-9778
FAX (852) 2722-1369

Japan
Atmel Japan K.K.
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
TEL (81) 3-3523-3551
FAX (81) 3-3523-7581

Atmel Operations

Atmel Colorado Springs
1150 E. Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906
TEL (719) 576-3300
FAX (719) 540-1759

Atmel Rousset
Zone Industrielle
13106 Rousset Cedex
France
TEL (33) 4-4253-6000
FAX (33) 4-4253-6001

Atmel Smart Card ICs
Scottish Enterprise Technology Park
East Kilbride, Scotland G75 0QR
TEL (44) 1355-803-000
FAX (44) 1355-242-743

Atmel Grenoble
Avenue de Rochepleine
BP 123
38521 Saint-Egreve Cedex
France
TEL (33) 4-7658-3000
FAX (33) 4-7658-3480

Atmel FPSLIC Hotline e-mail
fpslic@atmel.com

Atmel FPSLIC Hotline phone
1-(408) 436-4119

Fax-on-Demand
North America:
1-(800) 292-8635
International:
1-(408) 441-0732

e-mail
literature@atmel.com

Web Site
http://www.atmel.com

BBS
1-(408) 436-4309

1974A–06/01/xM