UART Programming

Overview

UART – *Universal Asynchronous Receiver-Transmitter* – protocol communication is common both historically and today. UART devices are standardized, with a common set of registers and typical programming procedure, so there are similarities even across platforms. Some registers are read-only, and others are write-only; these are often paired to use the name register number, with the proper register determined by the hardware based on the operation performed (read vs write).

UART Registers & Descriptions					
#	***Read Function***	***Write Function***	***#***	***Read Function***	***Write Function***
0	Receiver Buffer	Transmit Holding	4	Modem Control	
1	Interrupt Enable		5	Line Status	*reserved*
2	Interrupt ID	FIFO Control	6	Modem Status	*reserved*
3	Line Control		7	Scratch Pad	

We are principally concerned with the *Receiver Buffer Register* (RBR), which holds data that has been received (but not read), the *Transmit Holding Register* (**THR**), which holds the data to be sent until it is transmitted, and the *Line Status Register* (**LSR**), which provides the UART's status flags.

UART Registers & Descriptions					
#	***Hex***	***Description***	***#***	***Hex***	***Description***
0	`0x01`	Data available	4	`0x10`	Break signal received
1	`0x02`	Overrun error	5	`0x20`	THR is empty
2	`0x04`	Parity error	6	`0x40`	THR empty & line idle
3	`0x08`	Framing error	7	`0x80`	Erroneous data in FIFO

The LSR is an 8-bit register of flags. It can help us determine when the UART is ready to provide / receive more data. Before we send / receive, we need to check the register status:

```
# Receive message routine
def receive(buffer):
    while not buffer.is_full() and not buffer.complete_message:
       while not LSR & 0x01: pass # Wait for data to be available
        buffer.push_back(UART_RBR) # Read byte into buffer

# Transmit message routine
def tramsit(buffer):
    for byte in buffer:
        while not LSR & 0x20: pass # Wait for THR to be empty
       UART_THR = byte # Write the byte to the holding register
```

UART on x86

On x86 processors, UART devices are often referred to as "COM ports". The standard UART ports are COM1-COM4. The ports are accessed using the ***data bus*** – they are not memory mapped. To access a register, we add the register number to the COM Port base. To read the LSR of COM1 into r0, we would use a data bus instruction with the base and register offset:

COM Ports on x86	
Name	***Base Port***
COM1	`0x03F8`
COM2	`0x02F8`
COM3	`0x03E8`
COM4	`0x02E8`

```
in #0x3fd, r0           // 0x3f8 (base) + 5 (LSR reg. num) = 0x3fd
```

UART on ARM

On ARM processors, UART devices are typically memory mapped, so they are accessed using the typical memory space. Unfortunately, exactly where in the memory space differs based on device, but on Akita, it is located at `0x40100000`.

On ARM processors, each UART register is padded to 4 bytes. Here's how to read the LSR on Akita:

```
ldr r0, 0x40100014      // 0x40100000 (base) + 5 (LSR reg. num) * 4 (padding)
```