Next we will overview the specific UART functions on the TM4C microcontroller. This section is intended to supplement rather than replace the Texas Instruments manuals. When designing systems with any I/O module, you must also refer to the reference manual of your specific microcontroller. It is also good design practice to review the errata for your microcontroller to see if any quirks (mistakes) exist in your microcontroller that might apply to the system you are designing.

Tiva TM4C microcontrollers have eight UARTs. The specific port pins used to implement the UARTs vary from one chip to the next. To find which pins your microcontroller uses, you will need to consult its datasheet. Table 11.2 shows some of the registers for the UART0 and UART1. For the other UARTs, the register names will replace the 0 with a 1 – 7. For the exact register addresses, you should include the appropriate header file (e.g., **tm4c123gh6pm.h**). To activate a UART you will need to turn on the UART clock in the **RCGC1** register. You should also turn on the clock for the digital port in the **RCGC2** register. You need to enable the transmit and receive pins as digital signals. The alternative function for these pins must also be selected. In particular we set bits in both the AFSEL and PCTL registers.

The OE, BE, PE, and FE are error flags associated with the receiver. You can see these flags in two places: associated with each data byte in **UART0_DR_R** or as a separate error register in **UART0_RSR_R**. The overrun error (**OE**) is set if data has been lost because the input driver latency is too long. **BE** is a break error, meaning the other device has sent a break. **PE** is a parity error (however, we will not be using parity). The framing error (**FE**) will get set if the baud rates do not match. The software can clear these four error flags by writing any value to **UART0_RSR_R**.

The status of the two FIFOs can be seen in the **UART0_FR_R** register. The **BUSY** flag is set while the transmitter still has unsent bits, even if the transmitter is disabled. It will become zero when the transmit FIFO is empty and the last stop bit has been sent. If you implement busy-wait output by first outputting then waiting for **BUSY** to become 0 (right flowchart of Figure 11.10), then the routine will write new data and return after that particular data has been completely transmitted.

The **UART0_CTL_R** control register contains the bits that turn on the UART. **TXE** is the Transmitter Enable bit, and **RXE** is the Receiver Enable bit. We set **TXE**, **RXE**, and **UARTEN** equal to 1 in order to activate the UART device. However, we should clear **UARTEN** during the initialization sequence.

	31–12	11	10	9	8	7–0	Name
$4000.C000		OE	BE	PE	FE	DATA	UART0_DR_R

	31–3				3	2	1	0	
$4000.C004					OE	BE	PE	FE	UART0_RSR_R
	31–8	7	6	5	4	3	2–0		
$4000.C018		TXFE	RXFF	TXFF	RXFE	BUSY			UART0_FR_R
	31–16				15–0				
$4000.C024		DIVINT							UART0_IBRD_R
	31–6				5–0				
$4000.C028					DIVFRAC				UART0_FBRD_R
		7	6 – 5	4	3	2	1	0	
$4000.C02C		SPS	WPEN	FEN	STP2	EPS	PEN	BRK	UART0_LCRH_R
	31–10	9	8	7	6–3	2	1	0	
$4000.C030		RXE	TXE	LBE		SIRLP	SIREN	UARTEN	UART0_CTL_R
	31–12	11	10	9	8	7–0			
$4000.D000		OE	BE	PE	FE	DATA			UART1_DR_R
	31–3				3	2	1	0	
$4000.D004					OE	BE	PE	FE	UART1_RSR_R
	31–8	7	6	5	4	3	2–0		
$4000.D018		TXFE	RXFF	TXFF	RXFE	BUSY			UART1_FR_R

Help

	31–16				15–0				
$4000.D024		DIVINT							UART1_IBRD_R
	31–6						5–0		
$4000.D028					DIVFRAC				UART1_FBRD_R
	31–8	7	6 – 5	4	3	2	1	0	
$4000.D02C		SPS	WPEN	FEN	STP2	EPS	PEN	BRK	UART1_LCRH_R
	31–10	9	8	7	6–3	2	1	0	
$4000.D030		RXE	TXE	LBE		SIRLP	SIREN	UARTEN	UART1_CTL_R

Table 11.2. Some UART registers. Each register is 32 bits wide. Shaded bits are zero.

Help

The **IBRD** and **FBRD** registers specify the baud rate. The baud rate **divider** is a 22-bit binary fixed-point value with a resolution of 2^{-6}. The **Baud16** clock is created from the system bus clock, with a frequency of (Bus clock frequency)/**divider**. The baud rate is 16 times slower than **Baud16**

Baud rate = **Baud16**/**16** = (Bus clock frequency)/(16***divider**)

For example, if the bus clock is 80 MHz and the desired baud rate is 19200 bits/sec, then the **divider** should be 80,000,000/16/19200 or 260.4167. Let *m* be the integer part, without rounding. We store the integer part (*m*=260) in **IBRD**. For the fraction, we find an integer *n*, such that *n*/64 is about 0.4167. More simply, we multiply 0.4167*64 = 26.6688 and round to the closest integer, 27. We store this fraction part (*n*=27) in **FBRD**. We did approximate the divider, so it is interesting to determine the actual baud rate. Assume the bus clock is 80 MHz.

Baud rate = (80 MHz)/(16* (*m*+*n*/64)) = (80 MHz)/(16* (260+27/64)) = 19199.616 bits/sec

The baud rates in the transmitter and receiver must match within 5% for the channel to operate properly. The error for this example is 0.002%.

The three registers **LCRH**, **IBRD**, and **FBRD** form an internal 30-bit register. This internal register is only updated when a write operation to **LCRH** is performed, so any changes to the baud-rate divisor must be followed by a write to the **LCRH** register for the changes to take effect. Out of reset, both FIFOs are disabled and act as 1-byte-deep holding registers. The FIFOs are enabled by setting the **FEN** bit in **LCRH**.

CHECKPOINT 11.2

Assume the bus clock is 10 MHz. What is the baud rate if UART0_IBRD_R equals 2 and UART0_FBRD_R equals 32?

Hide Answer

divider = 0010.100000_2. or 2 plus 32/64 = 2.5. The baud rate is 10MHz/2.5/16 which is 250 kHz.

CHECKPOINT 11.3

Assume the bus clock is 50 MHz. What values should you put in UART0_IBRD_R and UART0_FBRD_R to make a baud rate of 38400 bits/sec?

Hide Answer

50,000,000/38400/16 is 81.3802, which is similar to 81 and 24/64. UART0_IBRD_R is 81 UART0_FBRD_R is 24. The baud rate is 50MHz/(81+24/64)/16 which is 38402 bits/sec.