Technical notes on using Analog Devices DSPs, processors and development tools
Visit our Web resources http://www.analog.com/ee-notes and http://www.analog.com/processors or e-mail processor.support@analog.com or processor.tools.support@analog.com for technical support.

UART Enhancements on ADSP-BF54x Blackfin® Processors

Contributed by Benno Kusstatscher

Rev 1 – November 6, 2007

Introduction

Compared to ADSP-BF52x, ADSP-BF53x, and ADSP-BF561 Blackfin® processors, the ADSP-BF54x series of Blackfin processors introduces new features to the UART module. This EE-Note summarizes the changes, describes their benefits, and assists in porting code.

The new features are grouped into the following categories:

- Automatic RTS/CTS hardware flow control
- Increased receive FIFO
- Finer bit rate granularity
- Programming model fits better into Blackfin architecture
- Improved interrupt processing
- Up to four UARTs (two with flow control)

This EE-Note briefly covers the topics above. This document assumes that the reader is familiar with the ADSP-BF53x UART module. For a complete description of the ADSP-BF54x UART, see the *ADSP-BF54x Blackfin Processor Hardware Reference*[1].

Programming Model

The UART on ADSP-BF53x processors was closely aligned to the industry-standard, 16450-compatible programming model. Besides the obvious advantage of code compatibility, this also had a down-side: the aged programming model did not adequately support the pipelined, throughput-optimized architecture of Blackfin processors.

No More Destructive Reads

The particular nature of the `UARTx_LSR` and `UARTx_IIR` registers with their destructive behavior on read operations required special attention to speculative read situations. Furthermore, it was almost impossible to cleanly separate the receive, transmit, and status handlers from each other.

The status bits that were read destructively have now been converted to write-one-to-clear (W1C) behavior. For example, the following `UARTx_LSR` read sequence cleared the frame error (`FE`) status bit implicitly:

```
if (*pUART0_LSR & THRE) { ... }
```

Now, the `FE` bit requires an explicit W1C clear operation:

```
*pUART0_LSR = FE;
```

No More UARTx_IIR Register

The `UARTx_IIR` register is obsolete. Due to its destructive read nature and its clumsy prioritization scheme, the `UARTx_IIR` register has been depreciated.

Interrupt service routines can determine the calling interrupt source by interrogating the `UARTx_LSR` and/or `UARTx_MSR` status registers.

For example, if a routine polled UARTx_IIR to determine whether there was a line error,

```
if (*pUART0_IIR == 0x6) { ... }
```

now it should poll the UARTx_LSR status register:

```
if (*pUART0_LSR & (OE|FE|PE|BI))
{
  *pUART0_LSR = OE|FE|PE|BI;
  ...
}
```

Interrupt prioritization can be done at the global level using interrupt assignment (SIC_IARx) registers.

UARTx_IER Set and Clear Register Pair

Often, it is required to temporarily disable UART receive, transmit, or status interrupts. Doing this on ADSP-BF53x processors required an "expensive" read-modify-write operation to the interrupt enable (UARTx_IER) register.

If the three interrupts were served by different service routines – especially with interrupt nesting enabled – special care was required to ensure that a read-modify-write operation in the transmit interrupt handler was not interrupted by the receive service routine. In many cases, there was no "cheaper" workaround other than disabling interrupts globally.

On the new UART, the UARTx_IER registers are replaced by a pair of registers (UARTx_IER_SET and UARTx_IER_CLEAR). Writing a '1' to the UARTx_IER_SET register enables an interrupt. Writing a '1' to UART_IER_CLEAR disables the respective interrupt. Writing '0' has no effect. Reading either register returns the enabled bits.

This new design permits separate interrupt handlers to toggle the associated interrupt enable bit independently without the risk of impacting each other.

Since the interrupt enable registers default to zero after reset, the initial setup instruction might be changed from

```
*pUART0_IER = ETBEI|ERBFI;
```

to simply

```
*pUART0_IER_SET = ETBEI|ERBFI;
```

if executed after reset.

When history is not known, instead execute:

```
*pUART0_IER_CLEAR = ~(ETBEI|ERBFI);
*pUART0_IER_SET = ETBEI|ERBFI;
```

If, however, any specific bit was required to toggle at runtime, the former code (when interrupt nesting is enabled)

```
short tmp;
/* temporarily enable */
tmp = cli();
*pUART0_IER |= ETBEI;
sti(tmp);
...
/* temporarily disable */
tmp = cli();
*pUART0_IER &= ~ETBEI;
sti(tmp);
```

can now be simplified to:

```
/* temporarily enable */
*pUART0_IER_SET = ETBEI;
...
/* temporarily disable */
*pUART0_IER_CLEAR = ETBEI;
```

No More MMR Address Sharing

On ADSP-BF53x processors, multiple memory-mapped registers (MMRs) shared the same address. The DLAB bit in the UARTx_LCR register functioned as an additional address bit to distinguish between accesses to the divisor latch registers (UARTx_DLH and UARTx_DLL) and the data registers (UARTx_RBR and UARTx_THR).

The DLAB bit is now obsolete. UARTx_DLH and UARTx_DLL have dedicated MMR addresses.

```
/* *pUART0_LCR = DLAB; */
*pUART0_DLL = divider & 0xFF;
```

```
*pUART0_DLH = (divider>>8) & 0xFF;
*pUART0_LCR = <new settings>;
```

Therefore, the UARTx_DLL and UARTx_DLH registers are accessible directly, and the first line of code in the above sequence can be deleted.

Interrupt Enhancements

Status Interrupt

Besides the regular transmit and receive interrupt request channels, each UART also features a status interrupt output. On ADSP-BF53x processors, the status interrupt was used only to signal UART line errors. On ADSP-BF54x processors, it is used for the following purposes:

- Line errors
- Alternate RX and TX signaling
- Transmit finished
- Modem status
- FIFO threshold

Alternate RX and TX Signaling

Since transmit and receive interrupts are routed through the DMA controller, they require that a DMA channel be assigned, even when DMA is not used. A disabled DMA channel still forwards a DMA request as a regular interrupt request to the System Interrupt Controller (SIC).

ADSP-BF54x processors have more interrupt-capable peripherals than DMA channels. Especially, UART2 and UART3 do not have DMA channels associated with them by default.

To avoid extensive use of a DMA channel for the express purpose of forwarding interrupts, the ADSP-BF54x UARTs have an option to redirect transmit and receive events to the status interrupt, which goes directly to the SIC controller. When the EGLSI bit in the global control register is set, transmit and receive events are not signaled to the regular transmit and receive interrupt channels; rather, they are signaled to the status interrupt channel, being ORed with regular status request.

This way, a UART module is still fully functional in non-DMA mode even without an associated DMA channel. There remains, however, the restriction that all UART events must be serviced by one common interrupt routine.

Transmit Finish Interrupts

On the transmit side, interrupt timing is an interesting topic. Normally, the UART transmitter issues an interrupt or DMA request when a UARTx_THR register is ready for new data, as signaled by the THRE bit. By this time, transmit data might be pending in the transmit shift register TSR.

Consequently, the transmit interrupt service routine is not yet permitted to disable the UART. Rather, the processor must still wait until the transmitter empty (TEMT) bit goes high.

On the ADSP-BF54x UART, the TEMT bit has been saddled with new interrupt functionality. There is a new sticky copy of the self-clearing TEMT bit in the UARTx_LSR register, called the transmit finished indicator (TFI) bit. If enabled by the ETFI bit, the TFI bit triggers a request to the status interrupt channel. The service routine must W1C the TFI bit. By that time, it is safe to disable the UART and EIA-485 style line drivers.

Alternatively, the transmit finish event can also be signaled through the DMA controller. This is enabled by the EDTPTI bit. The DMA controller informs the UART when the last data word of a work unit has been handed over to the UART. The next time the TEMT bit goes high, a special command is sent to the DMA channel. When DMAEN is also enabled, the DMA channel translates this command to an interrupt request and sends it to the SIC controller.

If the DMA works in stop mode, the UART's EDTPTI bit is set and the DI_EN bit is not set, then the normal DMA interrupt requested is delayed until the last data byte has left the UART. If both, EDTPTI and DI_EN are set, then two interrupts are requested on the same channel.

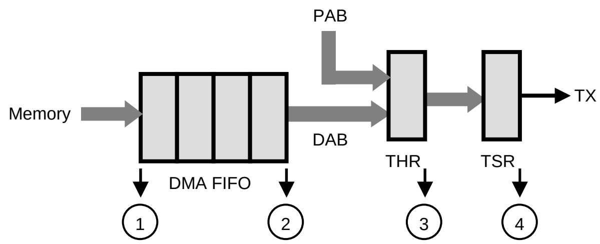

Figure 1. Transmit interrupt options

Figure 1 illustrates the resulting four interrupt options for UART transmit operation.

1. Regular TX DMA interrupt. By this time a new work unit can be initiated, but the DMA must not be disabled yet as data is still in the DMA FIFO.

2. TX DMA interrupt with the SYNC bit set. The DMA interrupt is delayed until all data has left the DMA FIFO. By this time, it is safe to disable and to re-assign the DMA channel.

3. Regular TX UART interrupt. In DMA mode, this request functions as DMA request and cannot generate interrupts. In non-DMA mode, the DMA channels forward this transmit request to the SIC controller. With the EGLSI bit set, this request goes to the status interrupt channel instead.

4. Transmission finished interrupt. At this time, all data has left the UART and it is safe to disable the UART module. If ETFI is enabled, a status interrupt is generated. If DMA is enabled and EDTPTI is set, the DMA controller requests an interrupt over the TX DMA channel. The user may choose to service the TFI event by the transmit or the status interrupt routine.

Receive FIFO

The ADSP-BF53x UARTs feature a two-stage receive buffer that consists of the receive buffer (UARTx_RBR) register and the serial sampling (RSR) register. The later is not accessible by software.

If enabled, a receive interrupt is issued when a data word has been copied from RSR to UARTx_RBR. This is performed at the time the first stop bit has been sampled. If software (or DMA) does not read the UARTx_RBR register before the receipt of stop bit of the second data word, the second word will overwrite the first word in UARTx_RBR and an overrun condition is reported. This means that software must respond within one data frame to avoid overrun errors.

On ADSP-BF54x processors, an additional 4-deep FIFO has been inserted between the RSR and UARTx_RBR registers. This is shown in Figure 2. Compared to the ADSP-BF53x implementation, the interrupt response time requirement has been relaxed by a factor of five. This feature does not require any software changes.

There is a status bit called RFCS in the newly introduced UARTx_MSR status register. It signals the receive FIFO's count status. The behavior of the bit depends on the new receive FIFO threshold (RFIT) bit in the UARTx_MCR register.

If RFIT=0, the RFCS bit indicates whether two or more data words are available in the receive FIFO. If RFIT=1, the RFCS bit indicates whether four or more data words are available in the receive FIFO. The RFCS bit clears automatically when the FIFO has drained below the watermark due to core or DMA reads from the UARTx_RBR register.

The enable receive FIFO count interrupt (ERFCI) bit enables interrupt signaling to the UART Status Interrupt channel based on the RFCS bit. In non-DMA mode, the interrupt service routine may rely on the status interrupt instead of the regular receive interrupt and reduce the interrupt load to one-half or one-fourth. Two (RFIT=0) or four (RFIT=1) bytes can be read from the receive buffer. When the RFCS bit clears, the interrupt request is also de-asserted.

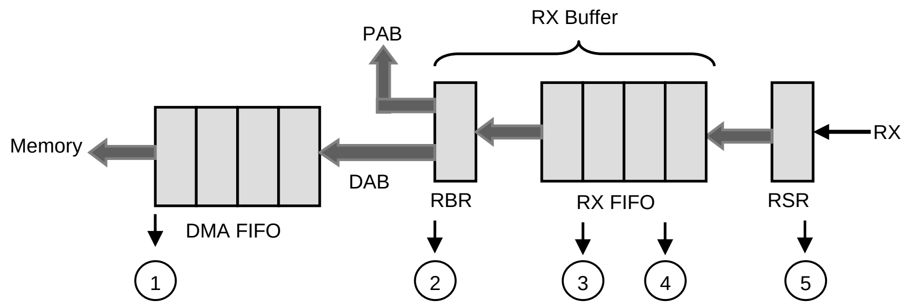

Figure 2. Receive interrupt options

Figure 2 illustrates the resulting five interrupt options for UART receive operation.

1. Regular RX DMA interrupt. By this time, all receive operation has been completed. Unless further data is in the receive pipe, it is safe to disable the DMA channel and/or the UART controller.
2. Normal interrupt in non-DMA mode when the ERBFI bit is set. All the receive status flags (DR, PE, FE, and BI) consistently report at this point of time. Consequently, the ELSI interrupt is also partly aligned here.
3. FIFO threshold interrupt option 1. When RFIT=0, the ERFCI interrupt signals that two bytes are ready in the UART receive buffer.
4. FIFO threshold interrupt option 2. When RFIT=1, the ERFCI interrupt signals that four bytes are ready in the UART receive buffer.
5. Overrun error. By the time the stop bit of the 6th word is received before the receive buffer has drained, an overrun error occurs and is reported to the ELSI interrupt.

Hardware Flow Control

Hardware flow control is a common handshake that enables a UART receiver to prevent the counterpart's transmitter from sending further data when the receive buffer may overflow. This feature is not available on UART0 and UART2.

The receiver generates an output signal, called $\overline{\text{RTS}}$ (request to send). It de-asserts the signal when the receive buffer is filled above a certain threshold. The transmitter senses to an input signal called $\overline{\text{CTS}}$ (clear to send). It stops transmission when $\overline{\text{CTS}}$ is de-asserted. However, it still cleanly finishes transmission of the currently processed word.

For bi-directional data flow, the hardware connection would look like Figure 3.

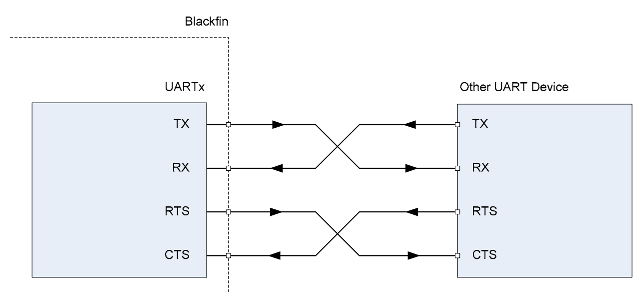

Figure 3. UART hardware flow control connection

In industry, the $\overline{\text{RTS}}$ and the $\overline{\text{CTS}}$ signals are usually active-low signals (send when low). Setting the FCPOL bit in the UARTx_MCR register inverts the signal polarity, and RTS and CTS become active-high signals (send when high).

$\overline{\text{RTS}}$ Generation of the Receiver

The ARTS bit in the UARTx_MCR register enables automatic generation of the $\overline{\text{RTS}}$ signal. If ARTS=0, the state of the $\overline{\text{RTS}}$ output signal is controlled by the MRTS bit manually. If, however, ARTS=1, the $\overline{\text{RTS}}$ output tells about the status of the receive buffer.

Similar to the RFIT bit, which controls the interrupt timing, the receive FIFO RTS threshold (RFRT) bit controls the assertion of the $\overline{\text{RTS}}$ signal. Unlike the interrupt, $\overline{\text{RTS}}$ assertion and de-assertion is subject to hysteresis.

$\overline{\text{RTS}}$ de-asserts if there are two (RFRT=0) or four (RFTR=1) bytes in the receive buffer and a third or fifth start bit is detected. $\overline{\text{RTS}}$ asserts again, when the receive buffer drains below the watermark of two (RFRT=0) or four (RFRT=1) due to UARTx_RBR reads again.

Since the $\overline{\text{RTS}}$ signals must be also enabled at port muxing level, it might be advantageous to have a pull-down resistor (pulling up if FCPOL=0) on this signal to prevent it from floating during the processor's reset cycling.

Also, a pull-up resistor on the TX output avoids floating signals while the processor is in reset.

$\overline{\text{CTS}}$ Sensing on the Transmitter

There is a new bit in the UARTx_MCR register called ACTS. If set, it enables automatic $\overline{\text{CTS}}$ sensing. Once the transmitter detects de-asserted level on the $\overline{\text{CTS}}$ input pin, it finishes transmission of the word currently in the TSR shift register. However, it prevents the word currently in the UARTx_THR register from being transferred to the TSR register until $\overline{\text{CTS}}$ is asserted again.

In addition, there is a new bit called CTS in the UARTx_MSR register that indicates assertion of the $\overline{\text{CTS}}$ input signal. There is also a sticky version of the CTS bit called SCTS, that can be cleared by W1C software operation.

If enabled by the EDDSI bit, a so-called modem status interrupt is signaled to the UART status interrupt, whenever SCTS =1.

Similar to the $\overline{\text{CTS}}$ input pin, software can also pause transmission. This is done by setting the XOFF bit in the UARTx_MCR register temporarily.

In loopback mode (LOOP_ENA=1), the receiver's $\overline{\text{RTS}}$ output is internally looped back to the transmitter's $\overline{\text{CTS}}$ input.

Bit Rate Generation

The UART module's base clock is derived from the system clock (SCLK) by dividing it by the 16-bit integer as represented by the UARTx_DLL and UARTx_DLH 8-bit registers.

For safety, the received UART data is oversampled 16 times. Thus, the effective bit rate is 16 times slower than the base clock.

If the UART is running at high bit rates, the granularity of the supported bit rates becomes very limited. For example, assume an SCLK frequency of 120 MHz and a nominal bit rate of 1 Mb/s. Then, a divisor value of 7 would result in a bit rate of 1071 Mb/s, which is off by 7%. A divisor value of 8 would result in 937500 b/s, which is not much better.

For this reason, the ADSP-BF54x UART features the new enable-divide-by-one (EDBO) bit in the global control register. With this bit set, the divide-by-16 circuit is bypassed. Consequently, the effective bit rate is:

$$bitrate = \frac{SCLK}{16^{1-EDBO} \times (DLH : DLL)}$$

With the `EDBO` bit set, the receiver still oversamples and makes one-out-of-three-samples majority decisions. However, the sample points in the middle of a bit are no longer generated by traditional synchronous logic.

In the above example, a divisor value of 120 would nicely generate the desired bit rate, when the `EDBO` bit is set.

Autobaud Detection

The principle of autobaud detection has not changed compared to the ADSP-BF53x UART. On ADSP-BF54x processors, each of the four UARTs has a GP timer associated. Its alternate capture input (`TACIx`) senses the UART receive input.

- TACI0 senses UART0 RX
- TACI1 senses UART1 RX
- TACI2 senses UART2 RX
- TACI3 senses UART3 RX

If the recommended autobaud pattern 0x40 is used and the timer is configured to capture periods between two falling edges, the `TIMERx_PERIOD` register will contain 8 bit times as shown in Figure 4.

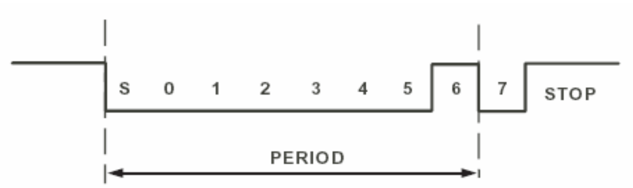

Figure 4. Autobaud detection using a 0x40 pattern

The UART clock divisor value is still calculated by shifting the resulting value in the `TIMERx_PERIOD` register seven bit positions to the right. If, however, the `EDBO` bit is set, the `TIMERx_PERIOD` value must only be shifted by three bit positions.

Code Porting Checklist

Porting code from ADSP-BF53x projects to ADSP-BF54x processors is usually quick. Nevertheless, a couple of code lines must be updated.

Required Changes

- Remove the `DLAB` instruction and access `UARTx_DLL` and `UART_DLH` directly.
- Replace expensive `UARTx_IER` read-modify-write sequences with W1S and W1C operations to the `UARTx_IER_SET` and `UARTx_IER_CLEAR` registers. If a `CLI`/`STI` instructions pair was used to protect against a read-modify-write, the `CLI`/`STI` instructions can be removed.
- Interrogate `UARTx_LSR` and/or `UARTx_MSR` instead of `UARTx_IIR`.

Recommended Changes

- Replace expensive `TEMT` polling loops with `TFI` interrupts
- In non-DMA mode, all three UART interrupt channels are serviced by the same routine. Set the `EGLSI` bit and free up DMA resources.
- In non-DMA mode, configure your receive interrupt routine to the `ERFCI` interrupt instead of the `ERBFI` interrupt to reduce interrupt load.
- If software was simulating flow control, enable automatic hardware flow control instead.
- If the bit rate provided by the UART clock divider is off the nominal bit rate, check whether setting the `EDBO` bit provides better transfer quality.

References

[1] *ADSP-BF54x Blackfin Processor Peripheral Hardware Reference.* Rev 0.3. October 2007. Analog Devices, Inc.

Document History

Revision	Description
Rev 1 – November 6, 2007 *by B. Kusstatscher*	Initial Release