KeyStone Training

Turbo Encoder Coprocessor (TCP3E)

Agenda

- Overview
- Architecture
- Usage
- Driver
- Summary
TCP3E Overview

• TCP3E = Turbo CoProcessor 3 Encoder
  – No previous versions, but came out at same time as third version of decoder co-processor (TCP3D)
  – Runs in parallel with DSP
• Performs Turbo Encoding for forward error correction of transmitted information (downlink for basestation)
  – Adds redundant data to transmitted message
  – Turbo Decoder in handset uses the redundant data to correct errors
  – Often avoids retransmission due to a noisy channel
Features Supported

- 3GPP, WiMAX and LTE encoding
  - 3GPP includes: WCDMA, HSDPA, and TD-SCDMA
- Code rate: 1/3
- Can achieve throughput of 250 Mbps in all three modes
- On-the-fly interleaver table generation
- Dual-encode engines with input and output memories for increased throughput
- Programmable input and output format within a 32-bit word
- Block sizes supported: 40 to 8192
- Tail biting for WiMAX
- CRC encoding for LTE

Architecture

- Overview
  - Architecture
- Usage
- Driver
- Summary
• Internally, TCP3E has dual (ping/pong) encode engines, config registers, input and output memories
• Externally, TCP3E looks like a single set of config regs and input / output buffers
• Routing to ping/pong is handled internally
• Alternates between ping and pong from one code block to the next
Interfaces

• VBUS config
  – Used for accessing the control and status registers
• VBUS DMA
  – Treated as a single interface from the point-of-view of memory map
  – Internally divided into separate write and read interfaces to improve throughput by allowing the transfer of data in and out simultaneously
• Events
  – TCP3E issues WEVTs to the EDMA to indicate that the input data to be encoded can be written by the EDMA
  – TCP3E issues REVTs to the EDMA to indicate that the encoded output data can be read by the EDMA

TCP3E Usage

• Overview
• Architecture
• Usage
  – Usage overview
  – Initialization
  – Configuration and data
  – EDMA setup
  – Error detection and interrupt
  – Debug and emulation support
• Driver
• Summary
Usage Overview and Dataflow

- Overview
- Architecture
- Usage
  - Usage overview
    - Initialization
    - Configuration and data
    - EDMA setup
    - Error detection and interrupt
    - Debug and emulation support
- Driver
- Summary

Process & Dataflow (1/3)

1. Initialization
   DSP initializes TCP3E mode, etc. via VBUS config interface after TCP3E is reset (hard or soft reset).

2. EDMA setup and input configuration register transfer
   DSP configures EDMA to transfer the input configuration registers, input data, and output data for a set of MCNT code blocks via the 32-bit DMA VBUS interface. DSP initiates config register transfer.

3. Input data transfer and start of encoding
   After input config registers are loaded, TCP3E issues a write event (WEVT) to EDMA to write the input data for the first code block to be encoded. TCP3E starts encoding after code block has finished loading.
Process & Dataflow (2/3)

4. Loading next code block
   If another encode engine is available (not currently in use) and there are more blocks to be encoded (based on MCNT), the TCP3E issues another write event (WEVT) to transfer the next code block’s input data. The TCP3E starts encoding the code block after it has finished loading.

5. Finished earlier code block
   When the earlier code block is finished encoding, the TCP3E issues a receive event (REVT) to EDMA that causes it to transfer the encoded block out of the TCP3E. If there are more blocks to be encoded (based on MCNT), go to previous step.

6. Finished set of MCNT code blocks
   After the entire set of MCNT code blocks has been encoded and retrieved, the EDMA can generate an interrupt to the DSP to signal that the encoding process is finished if desired.

7. Start next code block set
   If there is another set of code blocks to be encoded, the EDMA loads the next set of input configuration registers and the processing flow repeats.
Initialization

• Overview
• Architecture
• Usage
  – Usage overview and dataflow
  – Initialization
    – Configuration and data
    – EDMA setup
    – Error detection and interrupt
    – Debug and emulation support
• Driver
• Summary

Initialization Sequence

• Reset the TCP3E
  – Hard reset via chip-level register or external reset OR
  – Soft reset via TCP3E_SOFT_RESET register
• Program control registers
  – TCP3E_MODE
    • 3GPP, LTE, or WiMAX
  – TCP3E_EMU
    • FREERUN field determines if TCP3E halts on emulation suspend
    • SOFT field has no effect
  – TCP3E_ERR_MODE
    • Determines if TCP3E halts or continues running when specific error conditions are detected
Overview
Architecture
Usage
  - Usage overview and dataflow
  - Initialization
  - Configuration and data
    - EDMA setup
    - Error detection and interrupt
    - Debug and emulation support
Driver
Summary

### TCP3E_CFG0:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>R/W</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>11:0</td>
<td>MCNT</td>
<td>R/W</td>
<td>1 - 4095</td>
<td>Number of code blocks that need to be processed (setting value to 0 is illegal)</td>
</tr>
<tr>
<td>12</td>
<td>Reserved</td>
<td>R</td>
<td>0</td>
<td>Reserved (always reads 0)</td>
</tr>
<tr>
<td>26:13</td>
<td>BLOCK_SIZE</td>
<td>R/W</td>
<td>40 - 5114</td>
<td>Un-encoded code block size (in bits) (includes the 24-bit CRC for LTE when CRC generation is enabled)</td>
</tr>
<tr>
<td>28:27</td>
<td>IN_ORDER</td>
<td>R/W</td>
<td>0 - 3</td>
<td>Controls format conversion of input data:</td>
</tr>
<tr>
<td>30:29</td>
<td>OUT_ORDER</td>
<td>R/W</td>
<td>0 - 3</td>
<td>Controls format conversion of output data:</td>
</tr>
<tr>
<td>31</td>
<td>GEN_LTE_CRC</td>
<td>R/W</td>
<td>0 - 1</td>
<td>Only used for LTE mode to determine if CRC will be generated or not. This flag is ignored if not in LTE mode. Note that when this flag is enabled in LTE mode, the amount of input data transferred via EDMA will be 24 bits less than the specified BLOCK_SIZE.</td>
</tr>
</tbody>
</table>

WCDMA mode
LTE mode
WiMAX mode
### Input Configuration Register 1

TCP3E_CFG1

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>R/W</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15:0</td>
<td>INTLV_PAR0</td>
<td>R/W</td>
<td></td>
<td>3GPP mode: must be set to 0, LTE mode: f1, WiMAX mode: P0</td>
</tr>
<tr>
<td>31:16</td>
<td>INTLV_PAR1</td>
<td>R/W</td>
<td></td>
<td>3GPP mode: must be set to 0, LTE mode: f2, WiMAX mode: P1</td>
</tr>
</tbody>
</table>

### Input Configuration Register 2

TCP3E_CFG2

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>R/W</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15:0</td>
<td>INTLV_PAR2</td>
<td>R/W</td>
<td></td>
<td>3GPP mode: must be set to 0, LTE mode: must be set to 0, WiMAX mode: P2</td>
</tr>
<tr>
<td>31:16</td>
<td>INTLV_PAR3</td>
<td>R/W</td>
<td>-</td>
<td>3GPP mode: must be set to 0, LTE mode: must be set to 0, WiMAX mode: P3</td>
</tr>
</tbody>
</table>
Data Format

- The TCP3E input and output format is hard bits.
- If in LTE mode and CRC generation is enabled, then TCP3E will add CRC bits so number of input bits written by EDMA is BLOCK_SIZE - 24
- IN_ORDER and OUT_ORDER fields in the TCP3E_CFG0 register are set depending on the DSP endianness and data packing order in the system memory.
- Their effect is shown in the following diagram where b0 is the first data bit in time.

Bit Ordering Diagram

**LITTLE ENDIAN:**

<table>
<thead>
<tr>
<th>IN_ORDER</th>
<th>0</th>
<th>IN_ORDER</th>
<th>1</th>
<th>IN_ORDER</th>
<th>2</th>
<th>IN_ORDER</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address (Byte)</td>
<td>MSB</td>
<td>LSB</td>
<td>Address (Byte)</td>
<td>MSB</td>
<td>LSB</td>
<td>Address (Byte)</td>
<td>MSB</td>
</tr>
<tr>
<td>Base0</td>
<td>b3</td>
<td>b0</td>
<td>Base0</td>
<td>b4</td>
<td>b1</td>
<td>Base0</td>
<td>b5</td>
</tr>
<tr>
<td>Base1</td>
<td>b7</td>
<td>b4</td>
<td>Base1</td>
<td>b8</td>
<td>b5</td>
<td>Base1</td>
<td>b9</td>
</tr>
<tr>
<td>Base2</td>
<td>b11</td>
<td>b8</td>
<td>Base2</td>
<td>b12</td>
<td>b9</td>
<td>Base2</td>
<td>b13</td>
</tr>
<tr>
<td>Base3</td>
<td>b15</td>
<td>b12</td>
<td>Base3</td>
<td>b16</td>
<td>b13</td>
<td>Base3</td>
<td>b17</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**BIG ENDIAN:**

<table>
<thead>
<tr>
<th>IN_ORDER</th>
<th>0</th>
<th>IN_ORDER</th>
<th>1</th>
<th>IN_ORDER</th>
<th>2</th>
<th>IN_ORDER</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address (Byte)</td>
<td>MSB</td>
<td>LSB</td>
<td>Address (Byte)</td>
<td>MSB</td>
<td>LSB</td>
<td>Address (Byte)</td>
<td>MSB</td>
</tr>
<tr>
<td>Base0</td>
<td>b0</td>
<td>b3</td>
<td>Base0</td>
<td>b1</td>
<td>b4</td>
<td>Base0</td>
<td>b2</td>
</tr>
<tr>
<td>Base1</td>
<td>b5</td>
<td>b2</td>
<td>Base1</td>
<td>b6</td>
<td>b3</td>
<td>Base1</td>
<td>b7</td>
</tr>
<tr>
<td>Base2</td>
<td>b9</td>
<td>b6</td>
<td>Base2</td>
<td>b10</td>
<td>b7</td>
<td>Base2</td>
<td>b11</td>
</tr>
<tr>
<td>Base3</td>
<td>b13</td>
<td>b10</td>
<td>Base3</td>
<td>b14</td>
<td>b11</td>
<td>Base3</td>
<td>b15</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Tail Bits for 3GPP / LTE

- The TCP3E outputs three streams of data per code block.

\[ d_{k}^{(0)} = x_k \]
\[ d_{k}^{(1)} = z_k \]
\[ d_{k}^{(2)} = y_k' \]

for \( k = 0,1,2,...,K-1 \)

- Tail bits are packed at the end of three streams as:

\[ d_{k}^{(0)} = x_k \]
\[ d_{k}^{(1)} = z_k \]
\[ d_{k}^{(2)} = y_k' \]
\[ d_{k+1}^{(0)} = y_k' \]
\[ d_{k+1}^{(1)} = z_k \]
\[ d_{k+1}^{(2)} = x_k \]

EDMA Setup

- Overview
- Architecture
- Usage
  - Usage overview and dataflow
  - Initialization
  - Configuration and data
    - EDMA setup
      - Error detection and interrupt
      - Debug and emulation support
- Driver
- Summary
EDMA Setup

Three types of EDMA transfers are required:

1. Input configuration registers
   - Per code block set of MCNT code blocks
   - All registers must be written even if not used/changed
2. Input data (per code block)
3. Output data (per code block)

MCNT = 1 (Single Code Block) Example

- DSP initiates the process by setting the bit corresponding to EDMA channel X in the EDMA controller event set register.
- EDMA transfers configuration registers and links in the entry with the input data (info bits).
- TCP3E issues WEVT to initiate the input data transfer.
- TCP3E encodes data and issues REVT on channel Y, to initiate encoded data transfer to DSP.
- After the transfer completion, EDMA issues interrupt to DSP to signal the end of the encoding process.
MCNT = 5 Example

- Number of PaRAM entries is same as single code block
- PaRAM entries for input and output data are setup to move all five code blocks (one per WEVT/REVt)
Additional EDMA Info

- Other examples of EDMA programming can be found in the TCP3E User Guide:
  - Multiple code blocks of different size
  - Large number of code blocks
- PaRAM entry details for each of the three transfer types are also found in the TCP3E User Guide:
  - OPT (options field) settings
  - Source / destination address
  - A/B/C count parameters
  - Index parameters

Error Detection & Interrupt

- Overview
- Architecture
- Usage
  - Usage overview and dataflow
  - Initialization
  - Configuration and data
  - EDMA setup
    - Error detection and interrupt
      - Debug and emulation support
- Driver
- Summary
Error Detection

- TCP3E checks for several types of error conditions and can halt and/or generate an interrupt if they are detected.
- Error conditions include:
  - Programming errors (such as setting a control parameter in a register to an invalid value)
  - Incorrect DSP accesses to the TCP3E memories
- When a specific error condition is detected, the TCP3E will either halt or continue running based on the setting of the bits in the TCP3E_ERR_MODE register

<table>
<thead>
<tr>
<th>Bit Index in Registers</th>
<th>Error Condition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ILLEGAL_MODE</td>
<td>Mode is set to an illegal value (not 1, 2 or 4). This error condition may occur when writing to the TCP3E_MODE register or when the TCP3E starts a new encode operation.</td>
</tr>
<tr>
<td>1</td>
<td>MCNT_ZERO</td>
<td>MCNT is zero. This error condition may occur when writing to the TCP3E_CFG0 register.</td>
</tr>
<tr>
<td>2</td>
<td>BLKSZ_TOO_LARGE_OR_TOO_SMALL</td>
<td>Block size too large or too small. This error condition may occur when writing to the TCP3E_CFG0 register.</td>
</tr>
<tr>
<td>3</td>
<td>MODE_WRITE_DURING_ENCODE</td>
<td>Mode set during encode operation. This error condition may occur when writing to the TCP3E_MODE register.</td>
</tr>
<tr>
<td>4</td>
<td>INPUT_MEM_READ</td>
<td>Input buffer read in non-emulation mode. This error condition may occur when reading from the TCP3E input memory.</td>
</tr>
<tr>
<td>5</td>
<td>INPUT_BUFFER_WRITE_ACCESS_RANGE</td>
<td>Write to input buffer beyond current access range. This error condition may occur when writing to the TCP3E input memory.</td>
</tr>
<tr>
<td>6</td>
<td>OUTPUT_BUFFER_WRITE</td>
<td>Write to output buffer in non-emulation mode. This error condition may occur when writing to the TCP3E output memory.</td>
</tr>
<tr>
<td>7</td>
<td>OUTPUT_BUFFER_READ_ACCESS_RANGE</td>
<td>Read from output buffer beyond current access range. This error condition may occur when reading from the TCP3E output memory.</td>
</tr>
</tbody>
</table>
Error Interrupt and Recovery

• If an error condition has been enabled in the TCP3E_EINT_EN register, an interrupt will be generated when the error condition occurs.

• The following four steps should be followed when an error interrupt occurs:
  1. Identify the source of the interrupt by reading the TCP3E Error Interrupt Enabled Status register.
  2. If the error has not caused TCP3E to halt, the DSP clears the interrupt by writing to the TCP3E Error Interrupt Clear register.
  3. If the error has caused the TCP3E to halt, a soft reset must be issued at this time to clear the error source. There is no need to write to the TCP3E Error Interrupt Clear register. The EDMA should also be re-programmed.
  4. The DSP then writes an 8-bit vector to the TCP3E End of Interrupt register to acknowledge that it has serviced the TCP3E error interrupt condition.

Debug and Emulation Support

• Overview
• Architecture
• Usage
  – Usage overview and dataflow
  – Initialization
  – Configuration and data
  – EDMA setup
  – Error detection and interrupt
    – Debug and emulation support
• Driver
• Summary
Debug And Emulation Support

- TCP3E supports emulation suspend mode to enable debug, which forces a halt after all MCNT blocks in the current code block set are done encoding.
- In emulation suspend mode, the user can write to and read from every memory and register in the TCP3E.
- Transactions MUST be 32-bit aligned! (automatic if EDMA is used)
- The emulation suspend mode RAM memory map is different from the normal mode RAM mapping:
  - Since the selection is done internally during normal mode, the ping and pong memories are accessed at the same address.
  - Since both of the ping and pong memories have to be accessible in emulation suspend mode, separate memory addressing has to be used.
  - All memory accesses in emulation suspend mode will be performed as if the IN_ORDER and OUT_ORDER fields in the TCP3E_CFG0 register were set to ‘0’, meaning that no bit reversal or byte swapping will be performed.

TCP3E Driver

- Overview
- Architecture
- Usage
  - Usage overview and dataflow
  - Initialization
  - Configuration and data
  - EDMA setup
  - Error detection and interrupt
  - Debug and emulation support
- Driver
- Summary
TCP3E Driver Overview

• The TCP3E driver was created to abstract some of the hardware interface and optimize the code block scheduling to maximize throughput.

• Requirements:
  – BIOS 6.21
  – EDMA driver
  – QMSS driver

• Multiple input code block sets can be queued via Tcp3e_enqueue() function call.

• Driver can optionally write output data descriptor address to a QMSS queue as a notification mechanism.
TCP3E Driver Initialization

1. Initialize EDMA driver.
2. Optionally initialize QMSS driver, setup memory region, and allocate descriptors.
3. Setup TCP3E driver interrupt chain path within CP_INTC.
4. Open EDMA channels used by TCP3E driver and register magazine done ISR.
   - Use Tcp3e_getNumLinkedEdmaCh() to get number of linked channels required.
5. Allocate memory for the TCP3E driver instance.
   - Use Tcp3e_getNumBuf() to get number of memory buffers required.
   - Use Tcp3e_getBufDesc() to get memory buffer size, etc.
6. Call Tcp3e_init()

TCP3E Usage

1. Make sure driver is ready to accept more input blocks via Tcp3e_getFreeEntryCount()
2. Submit a list of code block sets via Tcp3e_enqueue()
3. Start TCP3E if not currently active via Tcp3e_start()
   - Use Tcp3e_getState() to check if active
4. Wait for completion of processing.
   - Magazine done interrupt
   - Optionally check QMSS queue if used (polling or accumulator interrupt)
For More Information

• For more information, please refer to the TCP3E User Guide.  
  http://www.ti.com/lit/SPRUGYS1

• For questions regarding topics covered in this training, visit the support forums at the 
  TI E2E Community website.