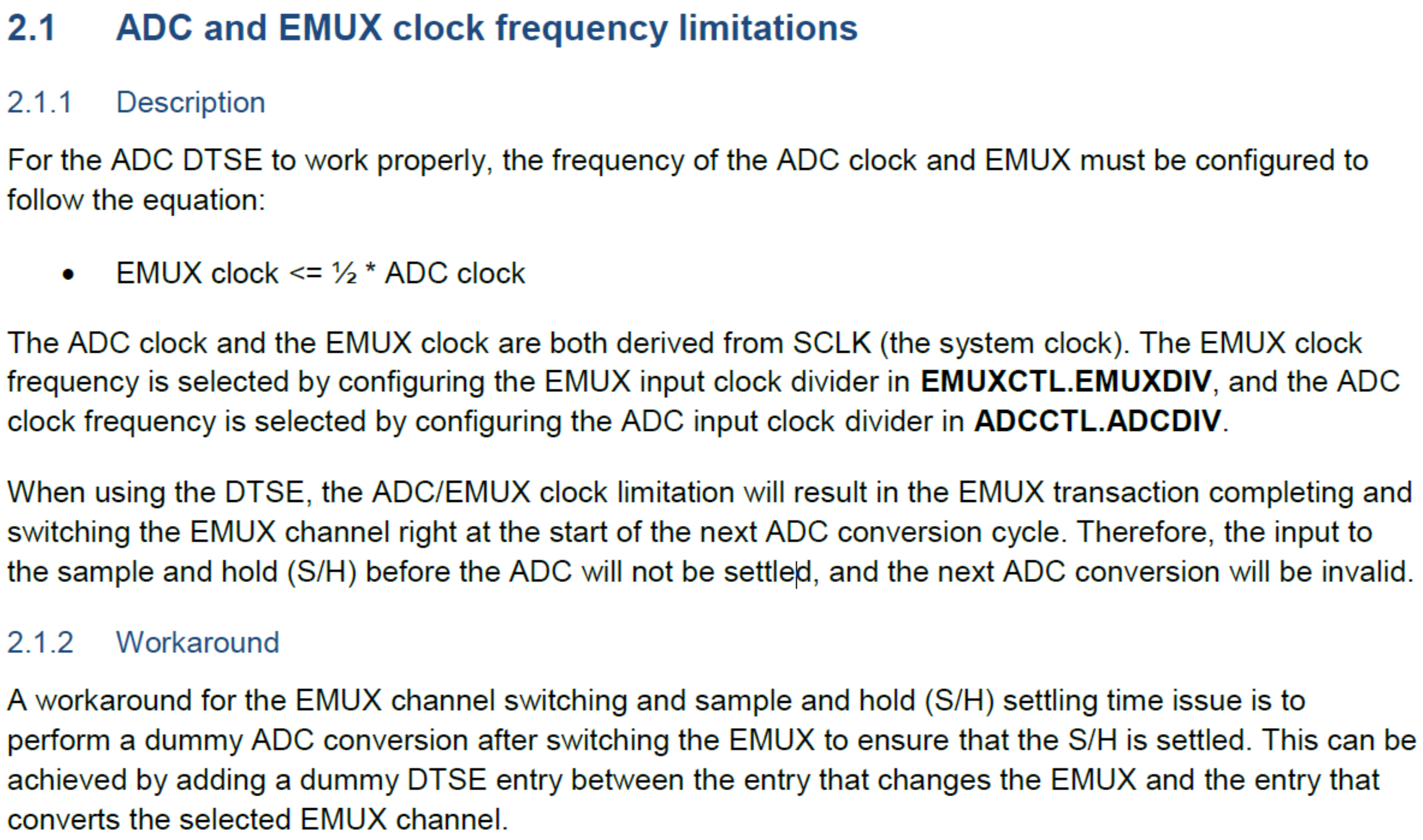
all around you

PAC55xx DTSE Workarounds

2019-08-28

PAC55xx DTSE Errata



2.1 ADC and EMUX clock frequency limitations

2.1.1 Description

For the ADC DTSE to work properly, the frequency of the ADC clock and EMUX must be configured to follow the equation:

- EMUX clock <= ½ * ADC clock

The ADC clock and the EMUX clock are both derived from SCLK (the system clock). The EMUX clock frequency is selected by configuring the EMUX input clock divider in **EMUXCTL.EMUXDIV**, and the ADC clock frequency is selected by configuring the ADC input clock divider in **ADCCTL.ADCDIV**.

When using the DTSE, the ADC/EMUX clock limitation will result in the EMUX transaction completing and switching the EMUX channel right at the start of the next ADC conversion cycle. Therefore, the input to the sample and hold (S/H) before the ADC will not be settled, and the next ADC conversion will be invalid.

2.1.2 Workaround

A workaround for the EMUX channel switching and sample and hold (S/H) settling time issue is to perform a dummy ADC conversion after switching the EMUX to ensure that the S/H is settled. This can be achieved by adding a dummy DTSE entry between the entry that changes the EMUX and the entry that converts the selected EMUX channel.

DTSE Errata Workaround Factors

- EMUX Clock Frequency <= ½ ADC Clock Frequency

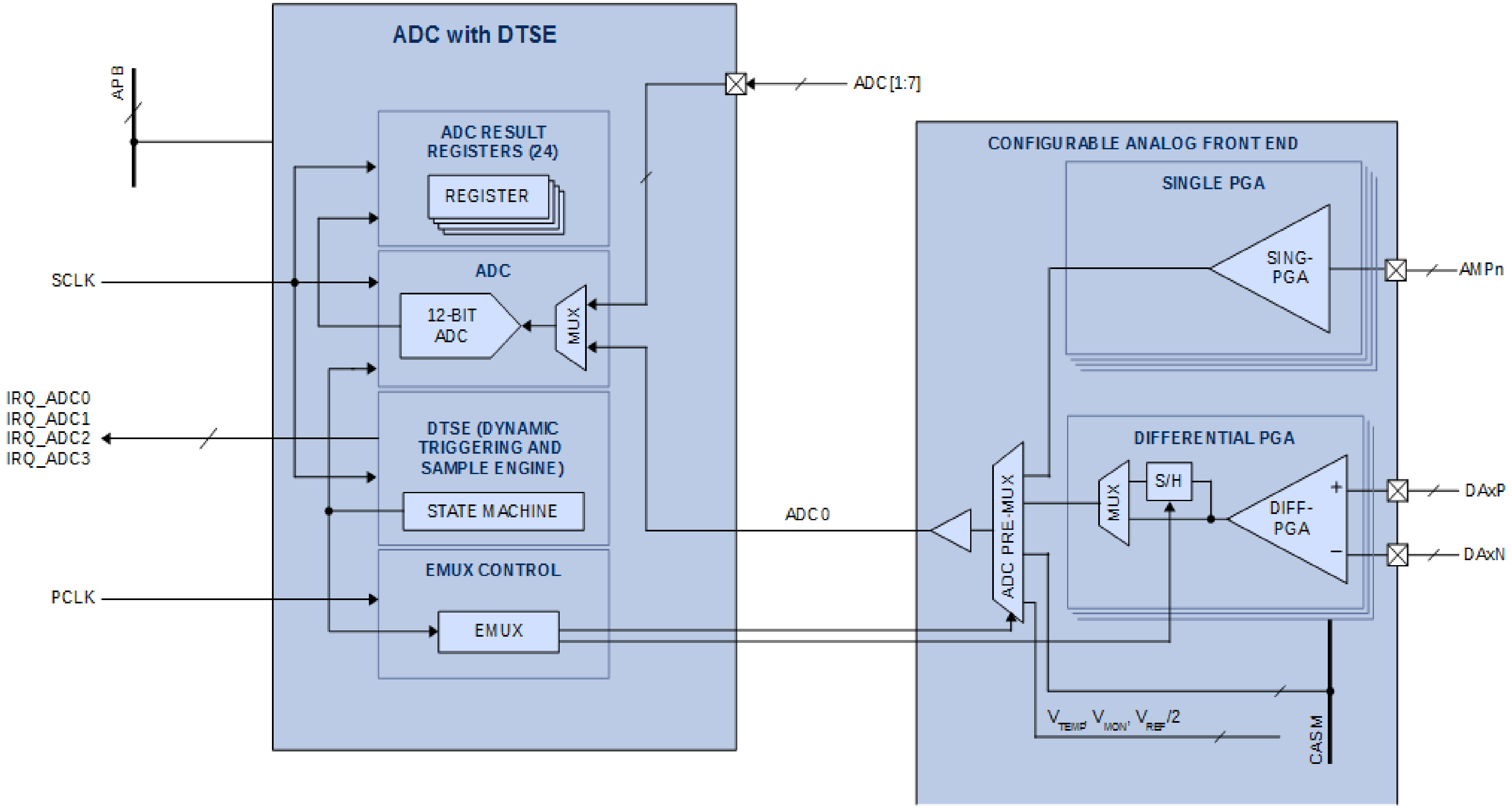
  If system clocks are set as

  - SCLK = PLL Clock = 300 MHz
  - HCLK (CPU Clock) = 150 MHz (maximum allowed M4 frequency)

  Then acceptable clock settings are:

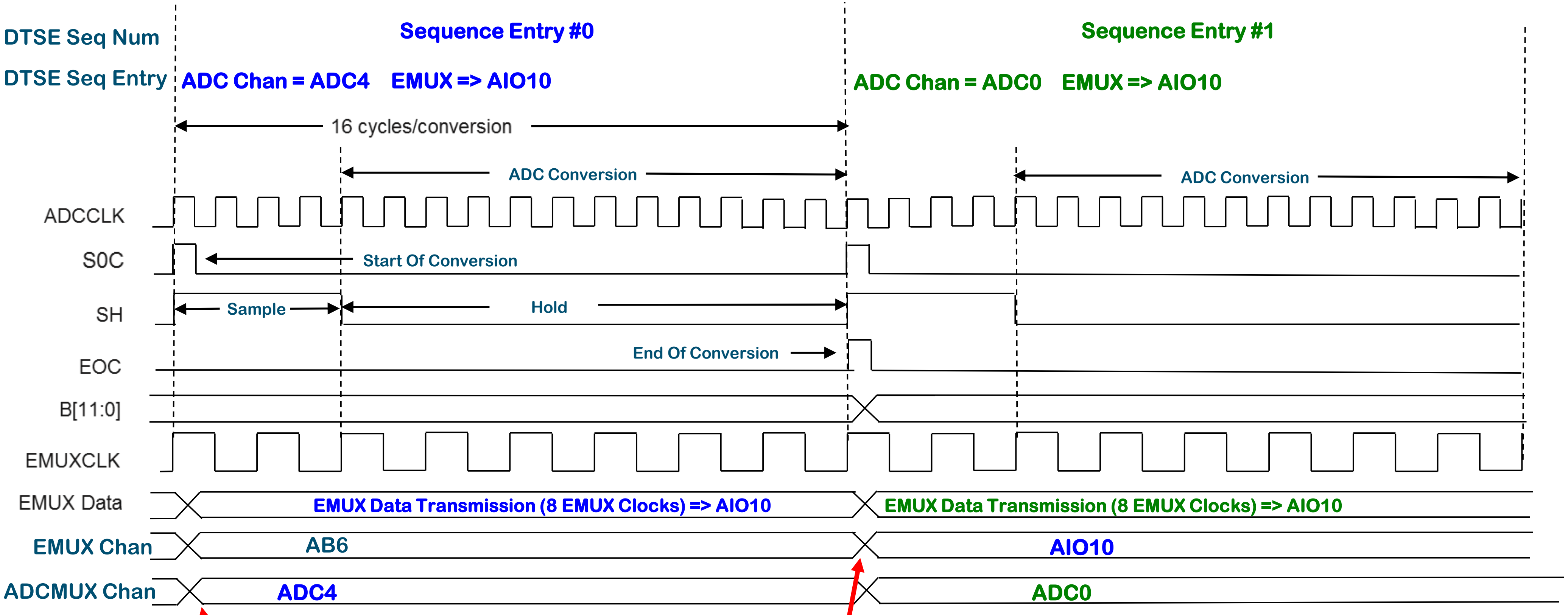
  - ADC Clock = 37.5 MHz = 1/8 * 300 MHz
  - EMUX Clock = 18.75 MHz = 1/16 * 300 MHz
- Slower EMUX clock results in EMUX Channel switching at end of ADC conversion cycle
- Therefore, a dummy conversion cycle must be inserted to allow the EMUX and ADC S&H voltage to settle.
- EMUXC field of DTSE Sequence Entries should always be set as
  - 01b: Send EMUX command before sample and hold

ADC and DTSE Block Diagram



EMUX Settling issues only come into play when converting ADC0

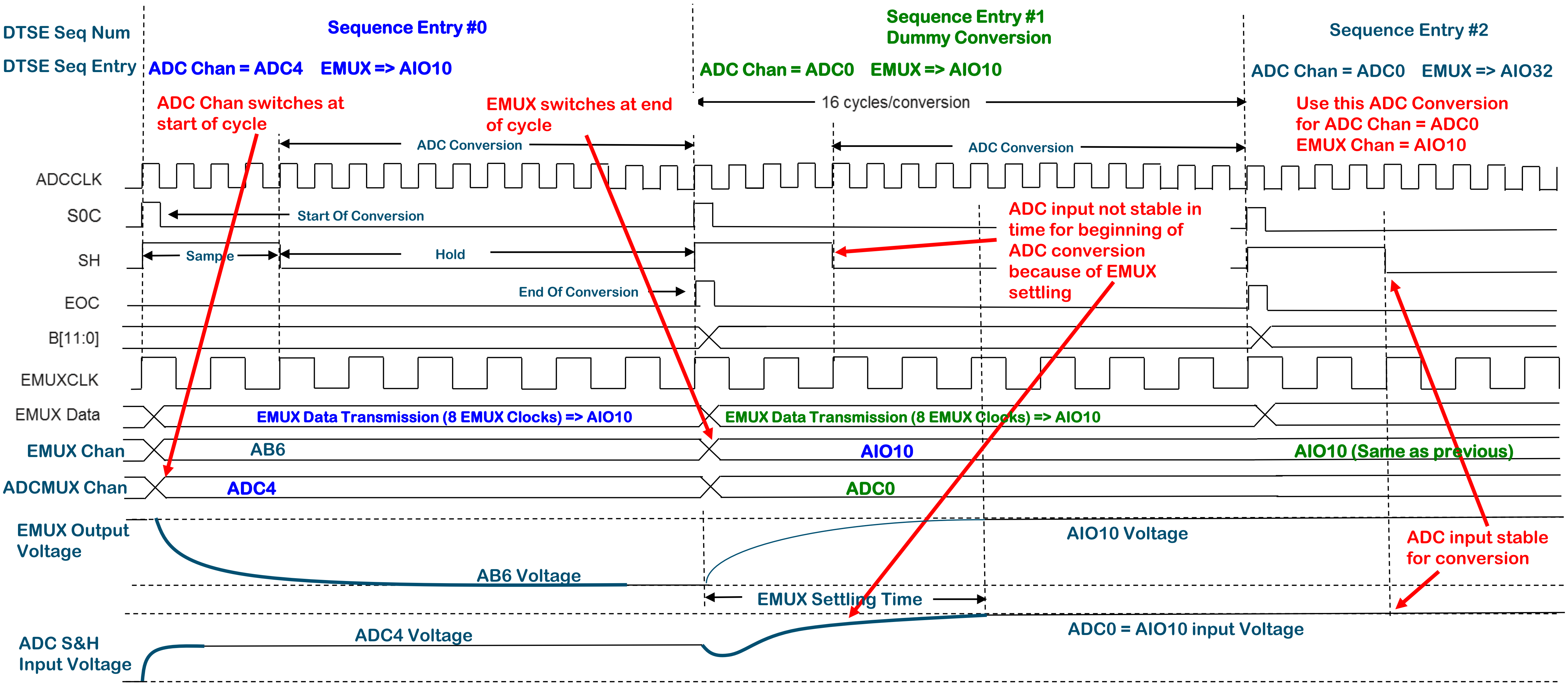
DTSE Sequence Timing Explained



ADC Chan switches at start of cycle

EMUX switches at end of cycle

DTSE ADC – EMUX Interactions



ADC Chan switches at start of cycle

EMUX switches at end of cycle

Use this ADC Conversion for ADC Chan = ADC0 EMUX Chan = AIO10

ADC input not stable in time for beginning of ADC conversion because of EMUX settling

ADC input stable for conversion

Sequence Example 1

EMUX set to AIO10 here for entry #2 conversion

```
//===== Setup DTSE Sequence =====
pac5xxx_dtse_seq_config(0, ADC0, EMUX_AIO10,  0,        0); //                        EMUX => AIO10
pac5xxx_dtse_seq_config(1, ADC0, EMUX_AIO10,  0,        0); // Dummy Entry            EMUX => same as previous entry
pac5xxx_dtse_seq_config(2, ADC0, EMUX_AIO32,  0,        0); // Convert ADC0 = AIO10;  EMUX => AIO32
pac5xxx_dtse_seq_config(3, ADC0, EMUX_AIO32,  0,        0); // Dummy Entry            EMUX => same as previous entry
pac5xxx_dtse_seq_config(4, ADC0, EMUX_AIO54,  0,        0); // Convert ADC0 = AIO32;  EMUX => AIO54
pac5xxx_dtse_seq_config(5, ADC0, EMUX_AIO54,  0,        0); // Dummy Entry            EMUX => same as previous entry
pac5xxx_dtse_seq_config(6, ADC0, 0         ,  0,        0); // Convert ADC0 = AIO54;  EMUX => Don't Care
pac5xxx_dtse_seq_config(7, ADC4, 0         ,  0,        0); // Convert ADC4           EMUX => Don't Care
pac5xxx_dtse_seq_config(8, ADC5, 0         ,  0,        0); // Convert ADC5           EMUX => Don't Care
pac5xxx_dtse_seq_config(9, ADC6, 0, ADC_IRQ0_EN, SEQ_END); // Convert ADC6           EMUX => Don't Care
```

Dummy entries not needed when ADC Channels other than ADC0 are converted

Last entry of this sequence

Entry 1 can't convert ADC0 with AIO10 input here. This is because the ADC S/H input is not settled in time following EMUX change from Entry 0. The EMUX changes just as Entry 0 conversion ends. ADC S/H needs more time to settle.

Interrupt at end of this entries conversion

```
//  Access Results
ADC0_AIO10 = PAC55XX_ADC->DTSERES2.VAL;
ADC0_AIO32 = PAC55XX_ADC->DTSERES4.VAL;
ADC0_AIO54 = PAC55XX_ADC->DTSERES6.VAL;
ADC4       = PAC55XX_ADC->DTSERES7.VAL;
ADC5       = PAC55XX_ADC->DTSERES8.VAL;
ADC6       = PAC55XX_ADC->DTSERES9.VAL;
```

Sequence Example 2 - Optimization

```
//===== Setup DTSE Sequence =====
pac5xxx_dtse_seq_config(0, ADC0, EMUX_AIO10,  0,        0); //                        EMUX => AIO10
pac5xxx_dtse_seq_config(1, ADC4, EMUX_AIO10,  0,        0); // Convert ADC4           EMUX => same as previous entry
pac5xxx_dtse_seq_config(2, ADC0, EMUX_AIO32,  0,        0); // Convert ADC0 = AIO10;  EMUX => AIO32
pac5xxx_dtse_seq_config(3, ADC5, EMUX_AIO32,  0,        0); // Convert ADC5           EMUX => same as previous entry
pac5xxx_dtse_seq_config(4, ADC0, EMUX_AIO54,  0,        0); // Convert ADC0 = AIO32;  EMUX => AIO54
pac5xxx_dtse_seq_config(5, ADC6, EMUX_AIO54,  0,        0); // Convert ADC6           EMUX => same as previous entry
pac5xxx_dtse_seq_config(6, ADC0, 0, ADC_IRQ0_EN, SEQ_END); // Convert ADC0 = AIO54;  EMUX => Don't Care
```

Non ADC0 channels
can be converted in
place of dummy entries

```
//  Access Results After Interrupt
ADC0_AIO10 = PAC55XX_ADC->DTSERES2.VAL;
ADC0_AIO32 = PAC55XX_ADC->DTSERES4.VAL;
ADC0_AIO54 = PAC55XX_ADC->DTSERES6.VAL;
ADC4       = PAC55XX_ADC->DTSERES1.VAL;
ADC5       = PAC55XX_ADC->DTSERES3.VAL;
ADC6       = PAC55XX_ADC->DTSERES5.VAL;
```

Sequence Example 3 – Single Sequence Optimization

```
//===== Setup DTSE Sequence =====
pac5xxx_dtse_seq_config(0, ADC0, EMUX_AIO32, 0,                0); // Convert ADC0 = AIO10;  EMUX => AIO32
pac5xxx_dtse_seq_config(1, ADC0, EMUX_AIO32, 0,                0); // Dummy Entry            EMUX => same as previous entry
pac5xxx_dtse_seq_config(2, ADC0, EMUX_AIO54, 0,                0); // Convert ADC0 = AIO32;  EMUX => AIO54
pac5xxx_dtse_seq_config(3, ADC0, EMUX_AIO54, 0,                0); // Dummy Entry            EMUX => same as previous entry
pac5xxx_dtse_seq_config(4, ADC0, EMUX_AIO10, ADC_IRQ0_EN,SEQ_END); // Convert ADC0 = AIO54;  EMUX => AIO10
```

EMUX set to AIO10 at end of Sequence

ADC0 = AIO10 converted at beginning of sequence;

settling occurs between triggers

This method works when the system uses only a single sequence

OR if all sequences in the system set EMUX to AIO10 during the last entry

```
//  Access Results After Interrupt
ADC0_AIO10 = PAC55XX_ADC->DTSERES0.VAL;
ADC0_AIO32 = PAC55XX_ADC->DTSERES2.VAL;
ADC0_AIO54 = PAC55XX_ADC->DTSERES4.VAL;
```

Sequence Example 4 – With All Diff Amps Held

```
//===== Setup DTSE Sequence =====
pac5xxx_dtse_seq_config(0, ADC0, EMUX_AIO10 | DIFFAMPS_HOLD, 0,       0); // Dummy Entry;           EMUX => AIO10 + HOLD
pac5xxx_dtse_seq_config(1, ADC0, EMUX_AIO32 | DIFFAMPS_HOLD, 0,       0); // Convert ADC0 = AIO10;  EMUX => AIO32 + HOLD
pac5xxx_dtse_seq_config(2, ADC0, EMUX_AIO32 | DIFFAMPS_HOLD, 0,       0); // Dummy Entry            EMUX => same as previous entry
pac5xxx_dtse_seq_config(3, ADC0, EMUX_AIO54 | DIFFAMPS_HOLD, 0,       0); // Convert ADC0 = AIO32;  EMUX => AIO54 + HOLD
pac5xxx_dtse_seq_config(4, ADC0, EMUX_AIO54 | DIFFAMPS_HOLD, 0,       0); // Dummy Entry            EMUX => same as previous entry
pac5xxx_dtse_seq_config(5, ADC0, EMUX_AIO10,       ADC_IRQ0_EN, SEQ_END); // Convert ADC0 = AIO32; EMUX => AIO6
```

S&H for AIO10, AIO32, and AIO54 Differential Amplifiers are held until the final conversion is complete.

EMUX set to AIO10 at end of Sequence
Hold Released

This method works when the system uses only a single sequence

OR if all sequences in the system set EMUX to AIO10 during the last entry

```
//  Access Results After Interrupt
ADC0_AIO10 = PAC55XX_ADC->DTSERES1.VAL;
ADC0_AIO32 = PAC55XX_ADC->DTSERES3.VAL;
ADC0_AIO54 = PAC55XX_ADC->DTSERES4.VAL;
```

Sequence Example 5 – Interrupt early

```
//===== Setup DTSE Sequence =====
pac5xxx_dtse_seq_config(0,  ADC0, EMUX_AIO10, 0,          0); // Convert ADC0 = AIO6   EMUX => AIO10
pac5xxx_dtse_seq_config(1,  ADC4, EMUX_AIO10, 0,          0); // Convert ADC4          EMUX => same as previous entry
pac5xxx_dtse_seq_config(2,  ADC0, EMUX_AIO32, 0,          0); // Convert ADC0 = AIO10; EMUX => AIO32
pac5xxx_dtse_seq_config(3,  ADC5, EMUX_AIO32, 0,          0); // Convert ADC5          EMUX => same as previous entry
pac5xxx_dtse_seq_config(4,  ADC0, EMUX_AIO54, 0,          0); // Convert ADC0 = AIO32; EMUX => AIO54 w/ Hold Diff Amps
pac5xxx_dtse_seq_config(5,  ADC6, EMUX_AIO54, 0,          0); // Convert ADC6          EMUX => same as previous entry
pac5xxx_dtse_seq_config(6,  ADC0, EMUX_AB7, ADC_IRQ0_EN, 0); // Convert ADC0 = AIO32; EMUX => Don't Care
pac5xxx_dtse_seq_config(7,  ADC0, EMUX_AB7,   0,          0); // Dummy                 EMUX => same as previous entry
pac5xxx_dtse_seq_config(8,  ADC0, EMUX_AB8,   0,          0); // Convert ADC0 = AIO7;  EMUX => Don't Care
pac5xxx_dtse_seq_config(9,  ADC0, EMUX_AB8,   0,          0); // Dummy                 EMUX => same as previous entry
pac5xxx_dtse_seq_config(10, ADC0, EMUX_AB9,   0,          0); // Convert ADC0 = AIO8;  EMUX => Don't Care
pac5xxx_dtse_seq_config(11, ADC0, EMUX_AB9,   0,          0); // Dummy                 EMUX => same as previous entry
pac5xxx_dtse_seq_config(12, ADC0, EMUX_AB6,   0,    SEQ_END); // Convert ADC0 = AIO9; EMUX => Don't Care
```

```
//  Access Results After Interrupt
ADC0_AIO6   = PAC55XX_ADC->DTSERES0.VAL;
ADC0_AIO10  = PAC55XX_ADC->DTSERES2.VAL;
ADC0_AIO32  = PAC55XX_ADC->DTSERES4.VAL;
ADC0_AIO54  = PAC55XX_ADC->DTSERES6.VAL;
ADC4        = PAC55XX_ADC->DTSERES1.VAL;
ADC5        = PAC55XX_ADC->DTSERES3.VAL;
ADC6        = PAC55XX_ADC->DTSERES5.VAL;

//Access after sufficient processing so that results are
guaranteed to be ready
ADC0_AIO7   = PAC55XX_ADC->DTSERES8.VAL;
ADC0_AIO8   = PAC55XX_ADC->DTSERES10.VAL;
ADC0_AIO9   = PAC55XX_ADC->DTSERES12.VAL;
```

Interrupt early! Interrupt will occur after this conversion. AIO10, AIO32, AIO54 can start to be processed earlier than end of sequence.

Thank You