*Project: RMC75E FPGA TEST BENCH*

*Module: Statemachine.vhd*

*Author: Satchel Hamilton*

*Company: Delta Motion*

*Date: 7/14/2023*

*Last updated: July 14, 2023*

The statemachine module controls the sampling of data from an A2D (Analog-to-Digital) converter.

The module takes several input signals, including:

* **SysReset**: A system reset signal or an indication that the PLL (Phase-Locked Loop) is not locked.
* **SysClk**: A 30 MHz system clock.
* **SlowEnable**: An enable signal that is active every 8th system clock (3.75 MHz).
* **SynchedTick**: A loop tick signal synchronized to the system clock.
* **LoopTime**: A 3-bit vector specifying the loop time duration (from 1/8 ms to 4 ms).

It provides the following output signals:

* **ExpA\_CS\_L**: A chip select signal for the ADC (Analog-to-Digital Converter).
* **ExpA\_CLK**: A clock signal for the ADC.
* **Serial2ParallelEN**: An enable signal for a shift register used for ADC output.
* **Serial2ParallelCLR**: A clear signal for the shift register used for ADC output.
* **WriteConversion**: An end-of-conversion indicator.

The module operates based on a state machine with various states defined by constants. Here are the state encodings:

* **StartState**: "000"
* **SampleHoldState**: "001"
* **ConvertState**: "011"
* **ConvertWaitState1**: "010"
* **ConvertWaitState2**: "110"
* **ConvertDoneState**: "111"
* **IncConvCountState**: "101"
* **InterConvDelayState**: "100"

The module uses delay times to spread the data samples over the control loop period. The delay values are based on the specified loop time. Here are the defined delay constants for different loop times:

* **eighth\_ms\_interconversion\_delay**: 5 counts
* **quarter\_ms\_interconversion\_delay**: 60 counts
* **half\_ms\_interconversion\_delay**: 180 counts
* **one\_ms\_interconversion\_delay**: 412 counts
* **two\_ms\_interconversion\_delay**: 881 counts
* **four\_ms\_interconversion\_delay**: 1820 counts

The module includes internal signals and registers to keep track of various conditions and timing. Here are some of the significant signals and registers:

* **State**: A 3-bit signal representing the current state of the state machine.
* **EndDelay**: A flag indicating whether the module needs to introduce a delay between conversions.
* **SampleHoldDone**: A flag indicating the completion of sample/hold operation.
* **ConversionDone**: A flag indicating the completion of the conversion process.
* **ConversionCounterEN**: A signal enabling the conversion counter.
* **ExpA\_CLK\_EN**: A signal enabling the ADC clock.
* **intExpA\_CLK**: A 2-bit vector representing the internal ADC clock.
* **ConversionCounter**: A 4-bit vector counting the number of conversions within a loop time.
* **CycleCounter**: A 5-bit vector used to track the logic sequence during individual conversions.
* **InterConversionDelayCNTR**: An 11-bit vector used to insert a delay between conversions.
* **InterConversionDelayEN**: A signal enabling the interconversion delay.
* **preInterConversionDelayTC**: A signal indicating whether the terminal count for interconversion delay is reached.
* **intConverting**: An internal signal indicating whether the module is currently converting.
* **Converting**: A synchronized version of **intConverting**.
* **intWriteConversion** and **intWriteConversion2**: Internal signals used to trigger data buffer writes.
* **intSerial2ParallelEN**: An internal signal indicating whether the converter data is being received.
* **WriteEN**: A signal indicating whether a write conversion is in progress.
* **asyncResetCycleCounter** and **ResetCycleCounter**: Signals used to reset the cycle counter.

Now, let's explain the behavior and timing of the module:

1. Initialization:
   * When the module receives a system reset signal (**SysReset = '1'**) or a synchronized tick (**SynchedTick = '1'**), it sets the state to the **StartState** to reset the state machine.
   * In the **StartState**, if the module is converting (**Converting = '1'**), the ADC chip select (**ExpA\_CS\_L**) is activated (**'0'**), and the shift register for ADC output (**Serial2ParallelCLR**) is cleared (**'0'**). The conversion counter (**ConversionCounterEN**) is disabled, and the state transitions to **SampleHoldState**.
   * If the module is not converting, the ADC chip select (**ExpA\_CS\_L**) is deactivated (**'1'**), the shift register for ADC output (**Serial2ParallelCLR**) is cleared (**'1'**), and the interconversion delay (**InterConversionDelayEN**) and conversion counter (**ConversionCounterEN**) are disabled.
2. Sample and Hold State (**SampleHoldState**):
   * In this state, the module waits until the sample/hold operation is complete (**SampleHoldDone = '1'**). It keeps the ADC chip select active (**'0'**).
3. Conversion State (**ConvertState**):
   * The module transitions to this state once the sample/hold operation is completed.
   * It checks if the conversion is done (**ConversionDone = '1'**). If not, it keeps the ADC chip select active (**'0'**) and enables the shift register for ADC data input (**intSerial2ParallelEN = '1'**).
   * If the conversion is done, the ADC chip select is turned off (**ExpA\_CS\_L = '1'**), the shift register for ADC data input is disabled (**intSerial2ParallelEN = '0'**), and the **intWriteConversion** signal is set to trigger a data write.
   * The state transitions to **ConvertWaitState1**.
4. Convert Wait State 1 (**ConvertWaitState1**):
   * In this state, the **intWriteConversion** signal is cleared, and the state transitions to **ConvertWaitState2**.
5. Convert Wait State 2 (**ConvertWaitState2**):
   * This state does not have any specific functionality and serves as a transition state.
   * The state transitions to **ConvertDoneState**.
6. Convert Done State (**ConvertDoneState**):
   * If the **EndDelay** flag is not set (**'0'**), the module enables the conversion counter (**ConversionCounterEN = '1'**) and transitions to **IncConvCountState**.
   * If the **EndDelay** flag is set (**'1'**), the shift register for ADC output (**Serial2ParallelCLR**) is cleared (**'1'**).
7. Increment Conversion Counter State (**IncConvCountState**):
   * After every conversion, there is a delay dictated by the loop time.
   * The conversion counter (**ConversionCounterEN**) is disabled, and the state transitions to **InterConvDelayState**.
8. Interconversion Delay State (**InterConvDelayState**):
   * This state introduces a delay between conversions based on the loop time.
   * When the delay is complete (**InterConversionDelayTC = '1'**), the interconversion delay is disabled, and the state transitions back to the **StartState**.
   * If the delay is not complete, the interconversion delay is enabled.
9. Other States:
   * In case the module is in an unrecognized state, it defaults to the **StartState** and resets the state machine.

Timing and Frequency:

* The **ExpA\_CLK** signal, which is the clock for the ADC, depends on the state and other conditions.
* During the **SampleHoldState** and **ConvertState**, the **ExpA\_CLK** signal is active every 8th system clock (**SlowEnable = '1'**).
* The **ExpA\_CLK** signal has a frequency of 3.75 MHz, corresponding to every 8th system clock.
* The **ExpA\_CLK** signal is toggled either positively or negatively based on the **SlowEnable** signal.

In conclusion, the module implements a state machine to control the sampling of data from an ADC. It performs sample/hold operations and conversions based on a specified loop time. The **ExpA\_CLK** signal exhibits a pattern where it is active every 8th system clock with a frequency of 3.75 MHz. The exact polarity of the clock (positive or negative) depends on the **SlowEnable** signal. The behavior and timing of the module ensure proper synchronization and sequencing of the ADC operations within the system.

Upon conducting a more thorough analysis of the provided code, here are some observations regarding the behavior of the clock signal (**ExpA\_CLK**) and the expected patterns:

1. Conversion Sequence: The code indicates that there will be four samples taken during each sample time, with two samples from channel 0 and two samples from channel 1. The conversion sequence is as follows: ch0 & ch1, ch0 & ch1 (wait 1 or 2 cycles), ch0 & ch1, ch0 & ch1 (wait...). This suggests that each channel is sampled twice before moving on to the next set of samples.
2. Loop Time and Interconversion Delay: The timing between sample groups is controlled by the **LoopTime** signal, which is a 3-bit vector representing the loop time. The code includes constants (**eighth\_ms**, **quarter\_ms**, **half\_ms**, etc.) that map specific loop time values to corresponding interconversion delay counts. The interconversion delay introduces a delay between conversions to spread the data samples over the entire control loop period.
3. Clock Enable and Pattern: The clock enable signal (**ExpA\_CLK\_EN**) is used to control the clock signal generation. It is active during specific conditions, such as during the SampleHoldState and ConvertState, depending on the state of the state machine and the SlowEnable signal. The clock pattern on **ExpA\_CLK** is generated by toggling the clock signal (**intExpA\_CLK**) based on the SlowEnable signal.
4. State Machine: The code includes a state machine that controls the sampling and conversion process. The state machine transitions between different states (**StartState**, **SampleHoldState**, **ConvertState**, etc.) based on the current state and various control signals. The clock behavior and patterns are influenced by the state transitions and the state machine's operation.
5. Delay Counts: The code provides constants (**eighth\_ms\_interconversion\_delay**, **quarter\_ms\_interconversion\_delay**, etc.) that specify the delay counts for different loop times. These values determine the duration of the interconversion delay between each set of samples.

The exact number of clock pulses on **ExpA\_CLK** and the regularity of the pattern would require a deeper understanding of the state machine's operation, signal assignments, and their interactions. Further analysis of the code, including the state machine transitions and the signal timing, would be necessary to determine the precise behavior of the clock signal and the expected patterns on **ExpA\_CLK**.

To gain a deeper understanding of the state machine's operation and determine the precise behavior of the clock signal (**ExpA\_CLK**) and the expected patterns, we can perform the following steps:

1. Review the State Machine Logic: Analyze the code and study the state machine's logic. Understand the different states (**StartState**, **SampleHoldState**, **ConvertState**, etc.) and their transitions. Identify the conditions under which the state machine changes states and how the clock signal generation is controlled.
2. Examine Signal Assignments: Look for signal assignments that affect the behavior of the clock signal. Pay attention to signals like **ExpA\_CLK\_EN**, **SlowEnable**, **intExpA\_CLK**, and any other signals involved in generating the clock signal. Understand how these signals interact and influence the clock generation process.
3. Investigate Timing Relationships: Explore the timing relationships between different signals and state transitions. Pay attention to signals such as **SynchedTick**, **SlowEnable**, and **InterConversionDelayEN** that control the timing and duration of the clock signal.
4. Trace Signal Paths: Follow the paths of important signals related to the clock signal and observe how they change throughout the state transitions. Identify any conditions or dependencies that affect the clock signal generation.
5. Simulate the Design: Utilize a hardware description language (HDL) simulator, such as ModelSim or Vivado Simulator, to simulate the code and observe the behavior of signals and the clock signal over time. By stepping through the simulation, you can gain insights into the clock pattern and validate your understanding of the code.
6. Debug and Refine: If necessary, adjust the code or modify the simulation setup to better analyze the clock signal behavior. Debug any unexpected issues or inconsistencies that you encounter during the simulation. Repeat the simulation process until you have a clear understanding of the clock signal's behavior.

"Interconversion delay" refers to a specific delay introduced between consecutive conversions in the analog-to-digital conversion process. It is a timing interval inserted to ensure that there is sufficient time between conversions, allowing for proper operation of the ADC and data processing.

The interconversion delay serves a few purposes:

1. Timing Separation: It provides a gap or delay between the completion of one conversion and the start of the next. This ensures that the ADC and associated circuitry have enough time to settle and stabilize before starting the next conversion. It helps prevent any potential interference or artifacts from previous conversions affecting the accuracy of subsequent conversions.
2. Signal Isolation: The interconversion delay helps isolate the signals and prevents any overlapping or interference between consecutive conversions. By introducing a controlled delay, it ensures that the data from one conversion is completely processed and separated before starting the next conversion.
3. Control Loop Timing: The duration of the interconversion delay is determined by the selected loop time. The loop time specifies the time period for a complete loop of conversions. The delay between conversions is adjusted to fill the remaining time within the loop after accounting for the actual conversion time.

In the code, different loop time values (eighth\_ms, quarter\_ms, half\_ms, etc.) are mapped to specific delay counts, which are used to control the duration of the interconversion delay. These delay counts are applied during the state transitions and timing control of the state machine, ensuring the proper timing relationships between conversions.

By carefully managing the interconversion delay, the code aims to ensure accurate and reliable ADC conversions without any signal interference or timing conflicts.

Comments on line 67 (statemachine):

-- The following delay times are used to spread the data samples over the entire control loop

-- period.

-- Each conversion takes 14us. There are 8 conversions per loop time so

-- 8 x 14us = 104us. 1ms - 104us = 896us. 896us is the total gap that

-- must be filled with 8 delay periods. 896us/8 = 112us/period.

-- The delay counter uses the SlowEnable clock enable line which runs at 3.75MHz

-- so, 112us/266.7ns = 419 clocks = 419 counts.

-- The same logic is applied to the 2ms loop time.

-- 2ms - 120us = 1.880ms.

-- 1.880ms/8 = 235us

-- 235us/266.7ns = 881 counts.

-- 500us loop time is 180 counts of interconversion delay

-- 250us loop time is 60 counts of interconversion delay

-- 125us loop time is 5 counts of interconversion delay

These comments provide explanations for the specific delay times used to control the interconversion delay in the code. Let's break down each comment to understand their meaning:

Comment 1:

The delay times mentioned are used to distribute the data samples evenly across the entire control loop period.

Each conversion takes 14 microseconds (us).

Within a loop time, there are 8 conversions.

Therefore, the total time consumed by the conversions is 8 x 14us = 104us.

The remaining time in the loop (1ms - 104us = 896us) needs to be filled with 8 delay periods.

To calculate the duration of each delay period, the total gap (896us) is divided by 8, resulting in 112us per period.

The delay counter in the code uses the SlowEnable clock enable line, which runs at a frequency of 3.75MHz.

To convert the delay duration to clock counts, the 112us is divided by the clock period of 266.7 nanoseconds (ns), resulting in 419 clocks or counts.

Comment 2:

The same logic is applied to the 2ms loop time.

The total loop time is 2ms, and a conversion takes 14us.

Subtracting the time consumed by conversions (8 x 14us = 112us) from the loop time gives 1.880ms.

Within this remaining time, there are 8 delay periods.

Dividing the remaining time (1.880ms) by 8 gives the duration of each delay period as 235us.

Similar to Comment 1, the delay duration is converted to clock counts, resulting in 881 counts using a clock period of 266.7ns.

Comment 3:

These comments provide specific delay counts for different loop time values.

For a 500us loop time, the interconversion delay consists of 180 counts.

For a 250us loop time, the interconversion delay consists of 60 counts.

For a 125us loop time, the interconversion delay consists of 5 counts.

Overall, these comments explain the calculations and mappings used to determine the duration and counts of the interconversion delay based on different loop time values. The delay periods ensure proper timing and separation between consecutive conversions to achieve accurate and reliable ADC operation.