Farrow-Based Decimating Sample Rate Converter

AN-347-1.0 Application Note

Introduction

The Farrow-based decimating sample rate converter demonstrates a Farrow resampler. The converter is supplied with the Altera® DSP Builder version 2.2.0. You can simulate its performance in MATLAB, change it as required for your application, generate a VHDL version and synthesize it to Altera devices. The converter is designed for an input clock rate identical to the system clock. For applications where the input rate is much lower than the system clock, time sharing should be implemented to get a cost effective solution. The DSP Buillder-example file is **FarrowResamp.mdl**.

For more information on DSP Builder, refer to the Altera website, **www.altera.com**.

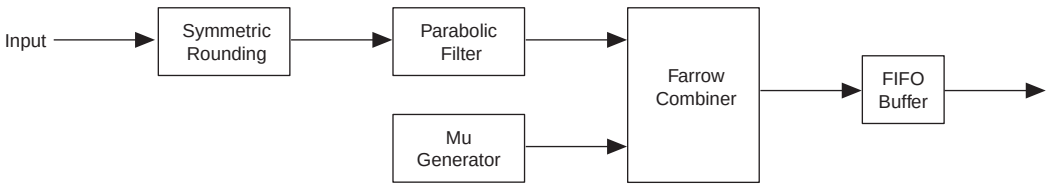
Background

Many integrated systems, such as software defined radios (SDR), require data to be resampled so that a unit can comply with different communication standards, where the sample rates are different. In some cases, where one clock rate is a simple integer multiple of another clock rate, resampling can be accomplished using interpolating and decimating FIR filters. However, in most cases the interpolation and decimation factors are so high that this approach is impractical. Farrow resamplers offer an efficient way to resample a data stream at a different sample rate. The underlying principle is that the phase difference between the current input and wanted output is determined on a sample by sample basis. This phase difference is then used to combine the phases of a polyphase filter in such a way that a sample for the wanted output phase is generated.

Functional Description

Figure 1 shows the Farrow-based decimating sample rate converter block diagram.

Figure 1. Block Diagram



The Farrow-based decimating sample rate converter comprises the following blocks:

- Symmetric rounding
- Parabolic filter
- µ generator
- Farrow combiner
- FIFO buffer

Symmetric Rounding

Symmetric rounding ensures that no DC offset is introduced through the rounding operation.

Parabolic Filter

The parabolic filter is a low-pass filter with a passband = 0.25 and a stopband = 0.75 . The filter coefficients for this polyphase filter are suitable for a parabolic farrow combiner, and have been calculated based on a low-pass filter with cutoff at 0.25, and a passband at 0.75. The filter coefficients are:

```
h = [[0 1 0 0]
[-0.37102969585002 -0.65524639874618 1.34475360125379
-0.37102969585000]
[0.37102969585001 -0.34475360125381 -0.34475360125381
0.37102969585001]]
```

For efficient hardware implementation it is desirable to have the minimum number of multiplications. The filter can be rewritten as:

```
h = [[0 1 0 0]
[-0.37102969585002 -1+0.34475360125381
1+0.34475360125379 -0.37102969585000]
[0.37102969585001 -0.34475360125381 -0.34475360125381
0.37102969585001]]
```

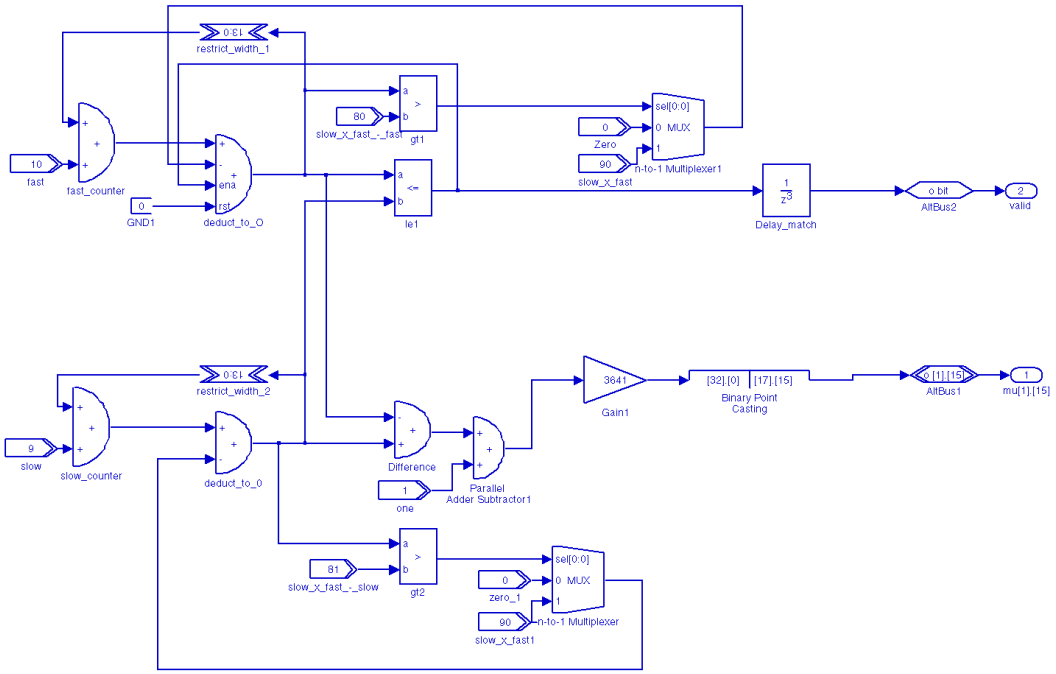
This way the filter can be implemented using just two multipliers. The samples are fed into both multipliers, and the output of the multipliers is fed into two delay chains. A third delay chain contains delayed versions of the samples.

Adders and subtractors are used to combine the elements of the three delay chains to perform the filtering operation.

µ Generator

The µ generator calculates the phase difference between the current input sample and the desired output sample and generates the values for µ. µ describes the normalised phase difference between the current input sample and the wanted output sample. Figure 2 shows the µ generator schematic.

Figure 2. µ ***Generator Schematic***



One way to generate the values for µ is to use two counters. Assume the clock ratios can be described as a ratio of `slow_clock/fast_clock`. The lower counter increments by `slow_clock` until it reaches the value `slow_clock` × `fast_clock` and then falls back to 0 again.

The second counter is similar. It increments by `fast_clock` until it reaches the value `slow_clock` × `fast_clock`. However, it only increments if its current value is smaller than the current value of the other counter.

If this is not the case, the counter pauses and flag the count value as invalid. Value μ can be generated by taking the difference between the count values.

Because the counting is done in integer logic, the difference must be scaled to produce the normalized value between 0 and 1. This scaling is done using a constant gain multiplier, which multiplies with the inverse of `slow_clock`.

Farrow Combiner

The Farrow combiner replaces the polyphase selector in traditional polyphase filters. It weighs the individual polyphases before adding them.

FIFO Buffer

The farrow combiner works at the input clock rate. This means it generates more samples than it should. The task of the FIFO is to remove invalid samples, and also allows the transfer between the clock domains. While the write side works at input clock rate, the read side works at output clock rate.

Testbench

This testbench provides the following three stimuli:

- The sinewave generator is useful to analyse the effect of the resampling on the frequency spectrum. You can change the frequency by double-clicking the **Sine Wave** block, and changing the **Samples per Period** parameter
- The impulse generator generates a single impulse and is 0 for the rest of the simulation. This is a useful tool for investigating the filter, which is part of the farrow resampler
- The addition of two sinewaves

Getting Started

This section involves the following steps:

1. System Requirements
2. Install the Design
3. Run the Design

4. Change the Stimuli

5. Synthesize the Design

System Requirements

The design requires the following hardware and software:

- A PC running the Windows 98/2000/XP software
- Quartus II version 4.0 or higher
- DSP Builder version 2.2.0
- MATLAB version 6.5
  - Simulink version 5.0.2
  - DSP Blockset version 5.0

Install the Design

The design is only shipped with DSP Builder version 2.2.0

Run the Design

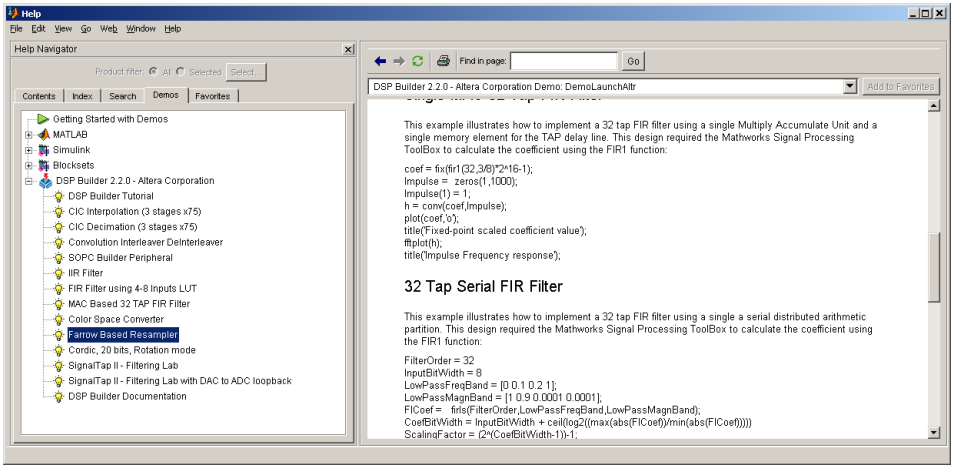
To run the design, follow these steps:

1. Open the MATLAB software.

2. At the command prompt, type `demo`.

3. Choose **DSP Builder version 2.2.0 Altera Corporation > Farrow Based Resampler** (see Figure 3).

Figure 3. Choose the Farrow-Based Resampler



4. At the bottom of the right-hand window, click **Run this Demo**.

5. Choose **Start** (Simulation menu).

Four waveforms are displayed, which show the spectrum at the following places:

- The input
- The input, quantized
- The input, if it had been just resampled by registering it in a flip-flop running at the output clock rate (the simplest way of changing sample rate)
- The output of the Farrow resampler

Change the Stimuli

You can change the stimuli by connecting the input to a different source. You can change the ratio of input clock to output clock, from the MATLAB command prompt, by setting the variables `slow_clock` and `fast_clock` to the required numbers. By default the variables are set to 9 and 10 respectively, to provide a ratio of 9/10, i.e., the output rate is 90% of the input clock rate. The variables `fast_clock` and `slow_clock` are used in a number of instances throughout the design. Use the following rules for setting the variables:

- The `slow_clock` output, i.e., the output clock, must be slower than `fast_clock` for a decimating Farrow filter

- `slow_clock` × `fast_clock` must be less than 2^{14}. This is a restriction of this particular implementation. You can work with ratios where this is not the case, but you then need to change the bit width in the accumulators in the mu_gen block
- If `slow_clock` is less than half of `fast_clock`, the filter works, but the results are sub-optimal. In this case it is better to double `slow_clock`, which should then still be less than `fast_clock`, and decimate at the output by rejecting every other sample

Synthesize the Design

You can generate VHDL and synthesize the same way as any other DSP Builder design using the **SignalCompiler** block.

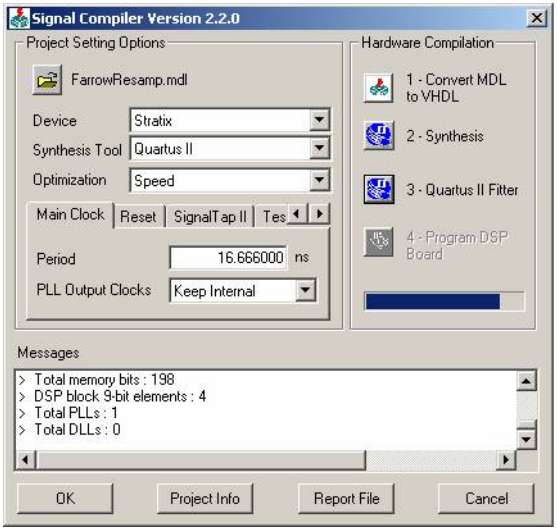
For more information on DSP Builder, refer to the *DSP Builder User Guide*.

To generate VHDL and synthesize, follow these steps:

You must simulate the design before you can synthesize the design.

1. Double click on the Signal Compiler icon in the schematic.
2. Click **Convert MDL to VHDL** (see Figure 4).

Figure 4. Signal Compiler



3. Click **Synthesis**.

4. Click **Quartus II Fitter**.

Performance

The converter requires the following resources to run at 79 MHz:

- 4 DSP block 9-bit elements
- 2 M512 RAM block
- 1 M4K RAM block
- 1,867 LEs
- 1 PLL