Quadratic interpolator

Theoretical background

Interpolation is a mathematical technique for estimating values between known data points. Polynomial approximations have the following form:

$$f(x) \approx \mathrm{a}_0 + \mathrm{a}_1 \mathrm{x} + \mathrm{a}_2 x^2 + \cdots + \mathrm{a}_{N-1} x^{N-1} \approx \sum_{i=0}^{N-1} \mathrm{a}_i x^i, \qquad x \in [x_{min}, x_{max})$$

where N is the number of terms in the polynomial approximation, $f(x)$ is the function to be approximated and a_i is the coefficient of the i-th term.

For a quadratic interpolator, N = 3, and the approximation is:

$$f(x) \approx \mathrm{a}_0 + \mathrm{a}_1 \mathrm{x} + \mathrm{a}_2 x^2$$

Quadratic interpolation provides a good balance between simplicity and accuracy — the polynomial is easy to evaluate in hardware, while still capturing the curvature of the underlying function much better than a simple linear approximation.

On the Moon systems must be lightweight, extremely reliable and power-efficient, with limited computational resources. Similarly, in FPGA design, we must make the most of limited resources like DSP blocks and logic elements. We chose a quadratic interpolator because it reflects the need to create a resource-efficient yet accurate approximation, which mirrors the engineering challenges faced in space missions.

Moreover, navigating or measuring conditions on the Moon often involves working with smooth, continuous data (like sensor readings or trajectories) where fast polynomial approximations are critical.

Goal

Your task is to perform quadratic interpolation of the trigonometric function:

$$f(x) = \sin\left(2x - \frac{\pi}{4}\right), \qquad x \in [0, 2)$$

based on the following diagram:

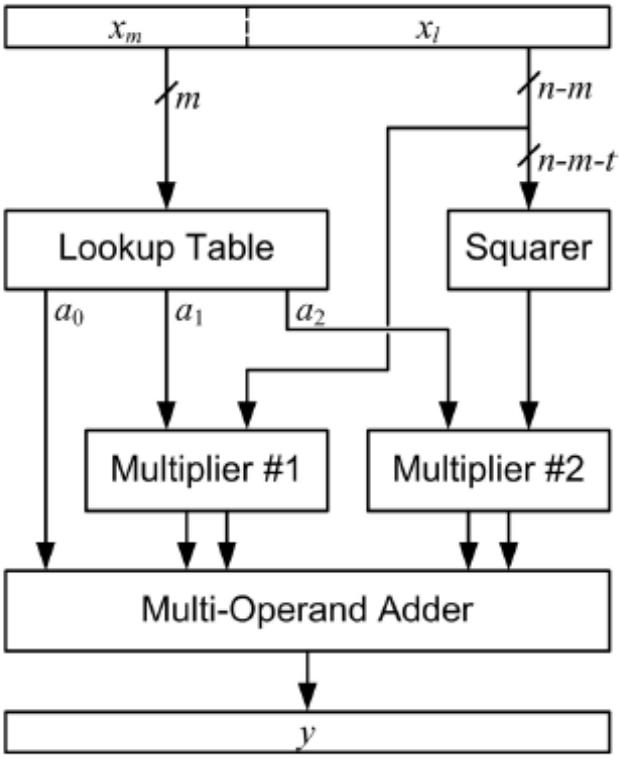


Figure 1. Quadratic interpolator block diagram.

$$y(x) = \mathrm{a}_0 + \mathrm{a}_1\mathrm{x} + \mathrm{a}_2 x^2$$

Figure 1 shows the block diagram of a quadratic interpolator, where $\mathrm{x_m}$ is used to select coefficients a_0, a_1 and a_2 from a lookup table. A specialized squarer computes ${x_l}^2$. Multiplier #1 computes $\mathrm{a}_1 \cdot \mathrm{x}_l$, and multiplier #2 computes $\mathrm{a}_2 \cdot {\mathrm{x}_l}^2$, which are then added to a_0 to produce the output.

(The diagram was taken from the publication "Optimized Linear, Quadratic and Cubic Interpolators for Elementary Function Hardware Implementations", the full document is available here: https://www.mdpi.com/2079-9292/5/2/17)

Phase I – Coefficient and test input generation

Understand the form of the function approximation:

$$y(x) = \mathrm{a} + \mathrm{b}\mathrm{x} + \mathrm{c}x^2$$

where:

- x - 24-bit unsigned fixed-point input **(u1.23)**
- y - 25-bit signed fixed-point output **(s2.23)**
- a, b, c - coefficients

In the src/tasks/task_10/resources folder, you will find supporting files: **coeff.m** and **generate_test_bins.m**.

Your task is to complete the provided coeff.m script (MATLAB), which calculates the coefficients for all 128 intervals in the range [0, 2), using the Taylor series approach. Store the generated coefficients in any format that is convenient for further processing. Just make sure that the resulting approximation meets the required accuracy specified in the "Evaluating the task" section.

Use the generate_test_bins.m script to generate input test values (x) for evaluation. The script allows you to choose whether to generate values located at the beginning, middle, or end of each interpolation segment. To make your choice, simply uncomment the appropriate line in the script. After generating the values, copy and paste them into the tb/task10.mem file. Do not rename the file - just overwrite its contents.

Optionally, you can generate your own set of x values. However, if you change the number of samples, you will also need to update the testbench (tb/tb_top.sv or tb/tb_lite.sv) accordingly to reflect the new input size:

```
50 | `elsif TASK_10
51 |     localparam NUM_SAMPLES_IN  = 128;
52 |     localparam NUM_SAMPLES_OUT = 128;
```

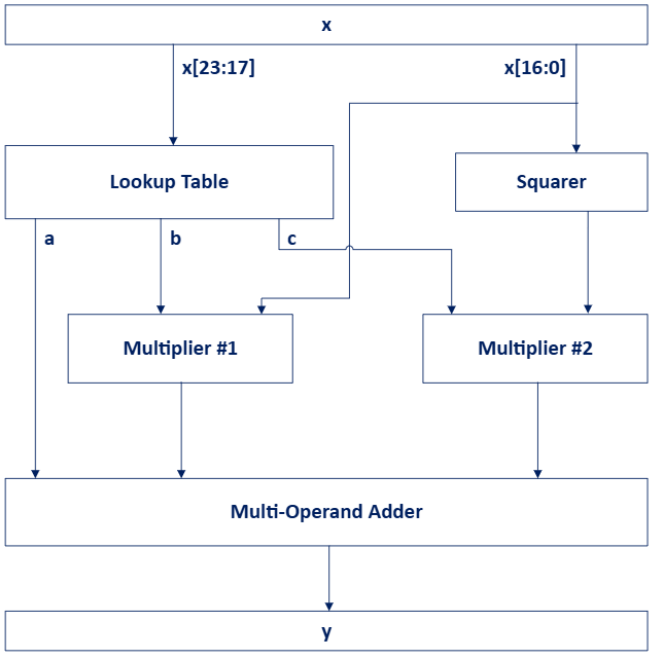
You can also adjust the **threshold** parameter in the testbench (tb/tb_top.sv or tb/tb_lite.sv) to test against a different accuracy target:

```
138
139     // CHANGE THRESHOLD HERE
140     static real THRESHOLD = 0.0000075;
141
```

Make sure not to exceed the maximum number of input samples allowed — see the "Input and output interfaces" section for detailed constraints.

Phase II - Hardware Implementation

Implement the interpolated function using the form: $y(x) = \mathrm{a} + \mathrm{bx} + \mathrm{c}x^2$. Use calculated coefficients a, b and c stored in a 128-entry lookup table. The appropriate entry is selected by 7 most significant bits of $x \rightarrow \boldsymbol{x}[\mathbf{23}:\mathbf{17}]$.



You can write HDL code or use MATLAB/Simulink with HDL Coder (a template - **Template_model_32.slx,** prepared by our partner ONT, is available in the src/tasks/task_10/resources folder). Make sure your design meets the required accuracy as specified in the "Evaluating the task" section.

Provided module

You can edit only the *task_10.sv* file. If you wish, you may add other files for supportive submodules.

The module has the following inputs and outputs:

Table 1 List of inputs and outputs of the task_10 module.

Signal name	Signal type	Bit length
i_clk	Input	1 bit
i_valid	Input	1 bit
i_rst	Input	1 bit
i_data	Input	32 bits
i_last	Input	1 bit
i_first	Input	1 bit
o_data	Output	32 bits
o_valid	Output	1 bit
o_last	Output	1 bit

IMPORTANT: The input is of type u1.23, so the 8 most significant bits (MSBs) in the input samples should be ignored. Similarly, the 7 MSBs in the output samples (of type s2.23) will be ignored by the judge during result checking.

Input and output interfaces

In one packet task receives **from 1 to 1000 samples** of input data, 32 bits each. The last sample in a packet is indicated by signal *i_last.* The task should send the same amount of output samples, 32 bits each, and should indicate the last sample in the output packet by providing signal *o_last.*

Task receives data in the way shown in the figure below:

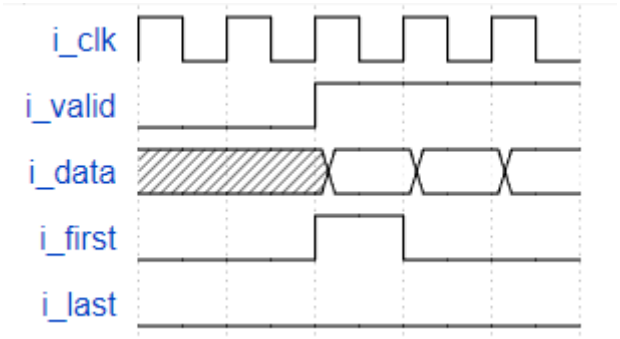


Figure 2 Input waveforms – the start of the packet.

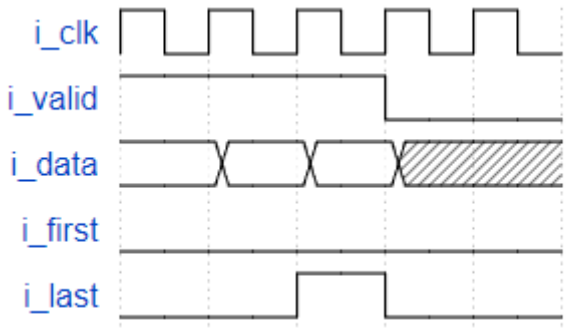


Figure 3 Input waveforms – the end of the packet.

To send data from the task in a proper way one needs to provide the *o_valid* signal that is synchronized with data on the *o_data* bus and o_last signal that is synchronized with the last sample in output packet.

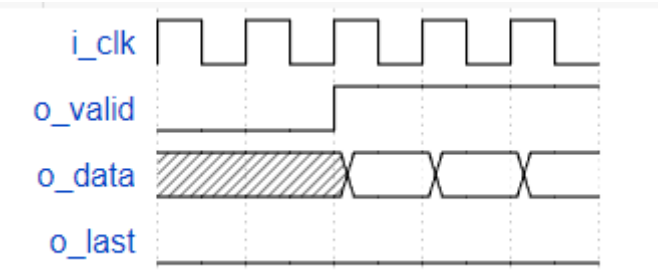


Figure 4 Output waveforms – the start of the packet.

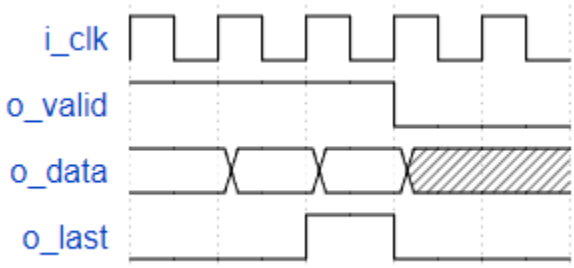


Figure 5 Output waveforms - the end of the packet.

Evaluating the task

To be awarded points, your output must meet the required level of approximation accuracy on all evaluated points. Points will be assigned based on the tightest accuracy threshold your implementation meets. The better the accuracy, the more points you receive.

$$\Delta = |f(x) - y(x)|$$

Threshold ($\Delta < \cdots$)	Points awarded
$7.5 \cdot 10^{-6}$	**7 points**
$7 \cdot 10^{-6}$	**14 points**
$6.5 \cdot 10^{-6}$	**21 points**

WARNING: Quadratic interpolation, like other polynomial interpolation methods, may produce results with varying accuracy depending on the value of x. The data used in Development Mode in the Judge are generated randomly, so passing this test does not guarantee that the interpolation function will achieve high accuracy for all possible input points.

In both **Development mode** and **Evaluation mode**, the task will be tested using a single set of data. In Development Mode, the test vector will be randomized on each run. In Evaluation Mode, the test vector will remain fixed across all runs and will be the same for every team (for more information, check chapter 2.3.1 **Testing modes** in the **FPGA_Hackathon_2025_Project_Guide** document). The scoring is provided in the table above.

Additional points may be awarded for **latency** (for more information, check chapter 2.3.2 **Bonus Points** in the **FPGA_Hackathon_2025_Project_Guide** document).