Sensors **2011**, *11*, 9160-9181; doi:10.3390/s111009160

OPEN ACCESS
sensors
ISSN 1424-8220
www.mdpi.com/journal/sensors

Article

Efficient Phase Unwrapping Architecture for Digital Holographic Microscopy

Wen-Jyi Hwang [1,⋆], Shih-Chang Cheng [1] and Chau-Jern Cheng [2,⋆]

[1] Department of Computer Science and Information Engineering, National Taiwan Normal University, Taipei, 117, Taiwan; E-Mail: jericho0703@gmail.com
[2] Institute of Electro-Optical Science and Technology, National Taiwan Normal University, Taipei, 117, Taiwan

[⋆] Author to whom correspondence should be addressed; E-Mails: whwang@csie.ntnu.edu.tw (W.-J.H.); cjcheng@ntnu.edu.tw (C.-J.C.).

Received: 4 August 2011; in revised form: 14 September 2011 / Accepted: 15 September 2011 / Published: 27 September 2011

Abstract: This paper presents a novel phase unwrapping architecture for accelerating the computational speed of digital holographic microscopy (DHM). A fast Fourier transform (FFT) based phase unwrapping algorithm providing a minimum squared error solution is adopted for hardware implementation because of its simplicity and robustness to noise. The proposed architecture is realized in a pipeline fashion to maximize throughput of the computation. Moreover, the number of hardware multipliers and dividers are minimized to reduce the hardware costs. The proposed architecture is used as a custom user logic in a system on programmable chip (SOPC) for physical performance measurement. Experimental results reveal that the proposed architecture is effective for expediting the computational speed while consuming low hardware resources for designing an embedded DHM system.

Keywords: phase unwrapping; digital holographic microscopy; FPGA; reconfigurable computing; system on programmable chip

1. Introduction

Digital holographic microscopy (DHM) [1,2] is a highly effective means for non-invasively capturing the amplitude and phase data of an optically transparent or reflective specimen [3,4]. Digital holograms that contain information about the specimen can be recorded and obtained using a charge-coupled device (CCD) or a complementary metal oxide semiconductor (CMOS). However, the phase map derived from the reconstructed image of a digital hologram is non-linearly wrapped lying in the interval ($-\pi$,π]. To obtain the three-dimensional profile of a specimen, the wrapped phase map must be unwrapped to a continuous phase [5]. Such a phase unwrapping procedure is also important in other applications, including synthetic aperture radar interferometry (InSAR) [6] and magnetic resonance imaging (MRI) [7]. The phase unwrapping process may be performed offline in some of these applications, whose primary concern is the quality of the unwrapped phases. By contrast, for the DHM or electronic speckle pattern interferometry applications, in addition to accurate unwrapped phase reconstructions, fast phase unwrapping operation is desired [8–12] for attaining realtime video rendering with high frame rates.

A simple raster scan algorithm is able to perform realtime phase unwrapping. However, in the presence of noise, the raster-scan algorithm may lead to an accumulation of error that eventually results in large deviations near the end of the accumulation. Popular approaches to the robust phase unwrapping include least square techniques, where the unwrapped phase is obtained as the function whose discrete gradient has the least squares deviation from its available estimate. The Poisson equations are then derived for the optimization, which can be solved by the preconditioned conjugate gradient (PCG) [13,14] and Gauss–Seidel techniques [13,15]. Recent advances in phase unwrapping include the $Z\pi M$ algorithm [16], which solves the problem as a sequence of binary optimization. Network programming techniques [16] and graph cut techniques [17] are then used for the optimization. Another efficient approach is to use branch cut technique for phase unwrapping, which can be implemented by hybrid genetic algorithms [18]. The common drawback of these techniques are that they are all iterative algorithms. The number of iterations may be high [19] and may vary [16,20] depending on the input wrapped phase map. For a realtime DHM, phase unwrapping algorithms with low and constant computational complexities are usually desired for providing fast video rendering with constant frame rate. Consequently, these robust phase unwrapping algorithms may make the design of a realtime DHM difficult. Moreover, for the embedded systems with limited computational capacities, the implementation of realtime robust phase unwrapping becomes a very challenging issue.

A number of implementations for fast phase unwrapping have been proposed [19–21]. The implementations presented in [20,21] are based on PCG [13,14] with field programmable gate array (FPGA) devices and graphic processing unit (GPU) [22] platforms, respectively. The implementation proposed in [19] is based on Gauss–Seidel techniques [13,15] with GPU platforms. Because both PCG and Gauss–Seidel techniques are iterative algorithms, only moderate acceleration is achieved. Moreover, some of these implementations are based on GPU, which may be difficult to be used for embedded devices due to high hardware cost and large power consumption.

The goal of this paper is to present a novel phase unwrapping hardware architecture for accelerating the computational speed of DHM. The algorithm [23] selected for the proposed architecture is also a least square technique for robust phase unwrapping. The algorithm is non-iterative, and therefore has constant computational complexity. As compared with its iterative least square counterparts [13–15], the selected algorithm is more computationally efficient [24,25]. In addition, it is based on fast Fourier transform (FFT), which can be efficiently implemented by hardware. Therefore, the algorithm is well-suited for the realtime DHM applications.

Based on the algorithm [23], the architecture is separated into four units: the pre-transform unit, the FFT unit, the post-transform unit, and the on-chip memory. The first three units are used for computation of the phase unwrapping algorithm. The on-chip memory is used for storing the source data and the intermediate results produced by these units so that the memory access time can be reduced. Novel pipeline architectures are proposed for the implementation of the pre-transform unit, the FFT unit, and the post-transform unit to maximize the throughput of the proposed architecture. Only a single multiplier and divider are used in the FFT unit and post-transform unit for lowering hardware resource utilization, respectively.

The proposed architecture has been implemented on FPGA devices so that it can operate in conjunction with a softcore CPU. Using the reconfigurable hardware, we are then able to construct a system on programmable chip (SOPC) system for the physical performance measurement for phase unwrapping in the embedded systems. As compared with its software counterpart running on Intel I-7 quad-core CPU, the proposed system has significantly lower computational time for phase unwrapping. In particular, when the image resolution is 513 $\times$ 513, the proposed system attains the speedup of 605 over its software counterpart. All these facts demonstrate the effectiveness of the proposed architecture.

2. The Phase Unwrapping Algorithm

This section briefly reviews the algorithm adopted for the hardware phase unwrapping implementation. Please refer to [23] for detailed discussion of the algorithm. Let $\zeta_{i,j}$ be the wrapped phase function of an unknown real-valued function $\phi_{i,j}$ for $0 \leq i \leq N$, and $0 \leq j \leq N$, where $-\pi < \zeta_{i,j} \leq \pi$, and $e^{j\zeta_{i,j}} = e^{j\phi_{i,j}}$. Let $\psi_{i,j}$ for $0 \leq i \leq 2N$, and $0 \leq j \leq 2N$ be the periodic extension of $\zeta_{i,j}$ using the mirror reflection technique. That is,

$$\psi_{i,j} = \begin{cases} \zeta_{i,j} & \text{for } 0 \leq i \leq N, 0 \leq j \leq N, \\ \zeta_{2N-i,j} & \text{for } N < i \leq 2N, 0 \leq j \leq N, \\ \zeta_{i,2N-j} & \text{for } 0 \leq i \leq N, N < j \leq 2N, \\ \zeta_{2N-i,2N-j} & \text{for } N < i \leq 2N, N < j \leq 2N. \end{cases} \tag{1}$$

Define

$$\Delta^x_{i,j} = \psi_{i+1,j} - \psi_{i,j}, \quad \Delta^y_{i,j} = \psi_{i,j+1} - \psi_{i,j} \tag{2}$$

for all i and j. Note that these values must be computed as *wrapped* phase differences, where the values 2π or -2π will be added as necessary to ensure that $\Delta^x_{i,j}$ and $\Delta^y_{i,j}$ lie in the interval $(-\pi, \pi]$.

Let $\bar{\phi}_{i,j}$ be an estimation of $\phi_{i,j}$ based on $\zeta_{i,j}$. The goal of the phase unwrapping algorithm is to find $\bar{\phi}_{i,j}$ minimizing

$$\sum_{i,j}(\bar{\phi}_{i+1,j} - \bar{\phi}_{i,j} - \Delta^x_{i,j})^2 + \sum_{i,j}(\bar{\phi}_{i,j+1} - \bar{\phi}_{i,j} - \Delta^y_{i,j})^2. \tag{3}$$

It can be shown that the optimal $\bar{\phi}_{i,j}$ is the solution to

$$(\bar{\phi}_{i+1,j} - 2\bar{\phi}_{i,j} + \bar{\phi}_{i-1,j}) + (\bar{\phi}_{i,j+1} - 2\bar{\phi}_{i,j} + \bar{\phi}_{i,j-1}) = \gamma_{i,j}, \tag{4}$$

where

$$\gamma_{i,j} = \Delta^x_{i,j} - \Delta^x_{i-1,j} + \Delta^y_{i,j} - \Delta^y_{i,j-1}. \tag{5}$$

Because both $\bar{\phi}_{i,j}$ and $\gamma_{i,j}$ are periodic, the Fourier transform can be used to solve Equation (4). Applying the 2N $\times$ 2N two-dimensional Fourier transforms to both sides of Equation (4) yields

$$\Phi_{m,n} = \Gamma_{m,n}/(2\cos(\pi m/M) + 2\cos(\pi n/N) - 4), \tag{6}$$

where $\Phi_{m,n}$ and $\Gamma_{m,n}$ are the Fourier transforms of $\bar{\phi}_{i,j}$ and $\gamma_{i,j}$, respectively. The function $\bar{\phi}_{i,j}$ is obtained by the inverse Fourier transform to Equation (6). The estimated $\phi_{i,j}$ is then obtained by restricting the results to the grid defined by $0 \le i \le N$, $0 \le j \le N$.

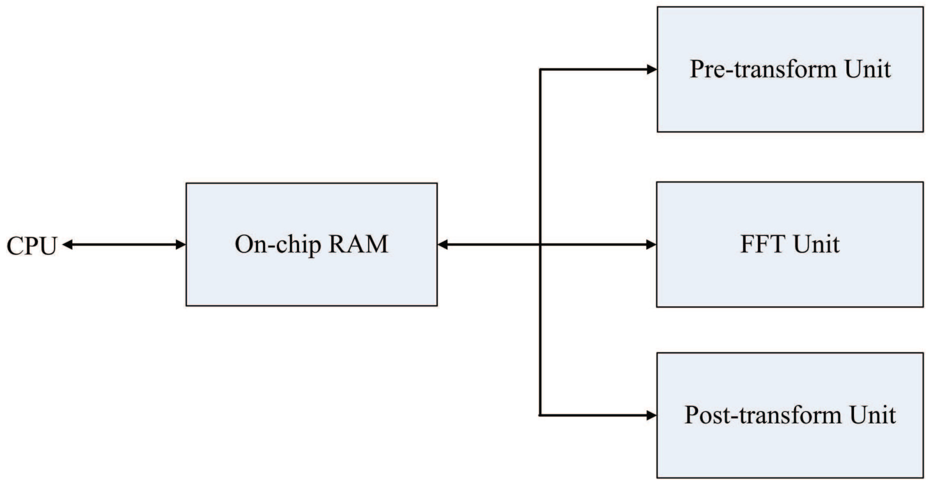
Based on the discussions shown above, the phase unwrapping algorithm using the FFT is summarized as follows:

```
Step 0:    Suppose $\zeta_{i,j}$, $0 \le i \le N$, $0 \le j \le N$, are given.
Step 1:    Compute $\gamma_{i,j}$, $0 \le i \le N$, $0 \le j \le N$, using Equation (5).
Step 2:    Compute $\Gamma_{m,n}$, $0 \le i < 2N$, $0 \le j < 2N$,
           using two-dimensional 2N $\times$ 2N fast Fourier transform (2D-FFT).
           The 2D-FFT operates as follows.
Step 2.1   For each row $i$ of the array $\gamma_{i,j}$,
           compute $\lambda_{i,j}$, $0 \le j < 2N$, using mirror reflection.
           That is, $\lambda_{i,j} = \gamma_{i,j}$ for $0 \le j \le N$, and $\lambda_{i,j} = \gamma_{i,2N-j}$ for $N \le j < 2N$.
Step 2.2   Compute $\Lambda_{i,n}$, $0 \le n < 2N$, the FFT of $\lambda_{i,j}$, $0 \le j < 2N$.
Step 2.3   Replace $i$-th row of the array $\gamma_{i,j}$ by $\Lambda_{i,n}$ with the restriction that $0 \le n < N$.
Step 2.4   After all of the rows are processed in this way,
           repeat the process (Step 2.1-2.3) on columns.
Step 3:    Compute $\Phi_{m,n}$ using Equation (6).
Step 4:    Compute the inverse FFT of $\Phi_{m,n}$ to obtain $\bar{\phi}_{i,j}$.
```

3. The Proposed Architecture

Figure 1 shows the proposed architecture for the FFT-based phase unwrapping algorithm. As shown in the figure, the proposed architecture can be separated into four units: the pre-transform unit, the FFT unit, the post-transform unit, and the on-chip memory. Given $\zeta_{i,j}$, $0 \le i \le N$, $0 \le j \le N$, the goal of the pre-transform unit is to compute $\gamma_{i,j}$. The FFT unit is then adopted for computing $\Gamma_{m,n}$. After that, the post-transform unit is used for calculating $\Phi_{m,n}$. Finally, the FFT unit is used again for computing $\bar{\phi}_{i,j}$ based on $\Phi_{m,n}$. The on-chip memory is used for storing both the original data and the intermediate and final results of the pre-transform unit, the FFT unit and the post-transform unit. Storing the original data and intermediate results in the on-chip memory effectively reduces the memory access time for the algorithm.

Figure 1. The proposed architecture for phase unwrapping.



3.1. On-Chip Memory

The on-chip memory consists of two identical RAM modules. Each RAM module is able to store an (N+1) $\times$ (N+1) array. The RAM modules are shared by all the units in the proposed architecture. They are used to store the original or intermediate results produced by each unit. These results will then be used as the source data for subsequent operations. The employment of the on-chip memory is able to significantly reduce the memory access time for phase unwrapping.

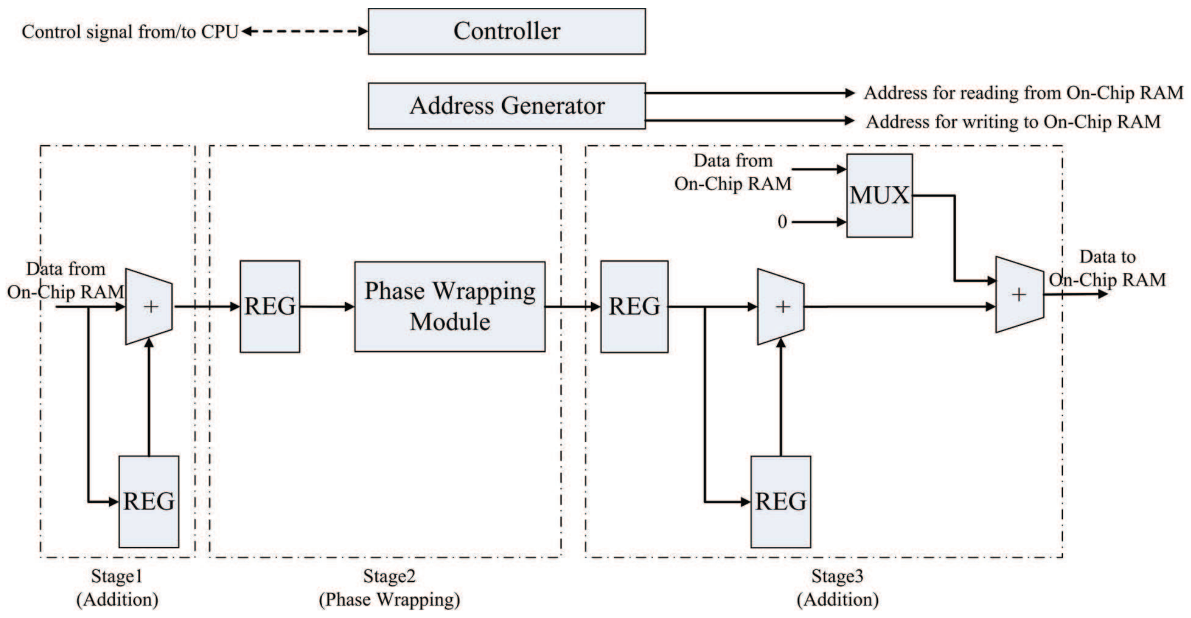
3.2. Pre-Transform Unit

The goal of the pre-transform unit is to implement Step 1 of the algorithm in hardware. Figure 2 shows the architecture of the pre-transform unit, which consists of controller, address generator, registers, adders, phase wrapping unit, and multiplexer. The address generator is used to generate addresses for reading the source data from the on-chip memory, and writing the results to the on-chip memory. The registers are used for the implementation of pipeline for enhancing the throughput. As shown in Figure 2, there are three stages in the pipeline. The first and the third stages are adders. The second stage is the phase wrapping unit, which is employed to ensure that the values of $\Delta^x_{i,j}$ and $\Delta^y_{i,j}$ lie in the interval $(-\pi, \pi]$.

Figure 3 depicts the architecture of the phase wrapping unit, which contains $2Q + 1$ modules. Each module i, $i = 1, ..., 2Q + 1$, contains two comparators for determining whether a phase difference computed by Equation (2) lies in the interval $(-(2Q + 1)\pi + 2(i - 1)\pi, -(2Q + 1)\pi + 2i\pi]$. The phase difference is first broadcasted to all the modules. After the interval in which the phase difference lies is identified, the phase wrapping operation is then performed accordingly so that the resulting wrapped phase difference lies in $(-\pi, \pi]$.

The source data for the pre-transform operations, $\zeta_{i,j}$, $0 \le i \le N$, $0 \le j \le N$, are stored in the on-chip RAM 1. The pre-transform unit then produces $\gamma_{i,j}$, $0 \le i \le N$, $0 \le j \le N$, and stores them in both the on-chip RAM 1 and RAM 2. The pre-transform unit computes $\gamma_{i,j}$ in two steps. At the first step, $\zeta_{i,j}$ is retrieved from the on-chip RAM 1 to compute $\Delta^x_{i,j} - \Delta^x_{i-1,j}$, which will then be stored in the on-chip RAM 2. Figure 4 shows the timing diagram for the pipeline operation of the pre-transform unit at the first step. Figure 5 reveals the input/output to each stage of the pipeline for the shaded time interval marked in the Figure 4.

Figure 2. The architecture of pre-transform unit, where REG and MUX are the abbreviations of register and multiplexer, respectively.



At the second step, $\zeta_{i,j}$ is retrieved again from the on-chip RAM 1 to compute $\Delta^y_{i,j} - \Delta^y_{i,j-1}$. In addition, $\Delta^x_{i,j} - \Delta^x_{i-1,j}$ is retrieved from the on-chip RAM 2. The summation of $(\Delta^y_{i,j} - \Delta^y_{i,j-1})$ and $(\Delta^x_{i,j} - \Delta^x_{i-1,j})$ forms $\gamma_{i,j}$, which will then be stored back to the on-chip RAM 1 and RAM 2 for the subsequent FFT operations. Figure 6 shows the timing diagram for the pipeline operation of the pre-transform unit at the second step. The input/output to each stage of the pipeline for the shaded time interval marked in the Figure 6 is then depicted in Figure 7. Note that, at the final stage of the pipeline, the MSBs (most significant bits) and LSBs (least significant bits) of $\gamma_{i,j}$ are stored in on-chip RAM 1 and RAM 2, respectively. By storing the results to two modules, the computation precision is then doubled for subsequent operations.

Note that, as shown in Figures 5 and 7, the input to the pre-transform circuit is $\psi_{i,j}$, which is the mirror reflected version of $\zeta_{i,j}$ in accordance with Equation (1). From Equations (2)–(5), it follows that the computation of $\gamma_{i,j}$ is based on $\psi_{i,j}$ instead of $\zeta_{i,j}$. When $0 \leq i \leq N$, $0 \leq j \leq N$, because $\psi_{i,j} = \zeta_{i,j}$, the $\zeta_{i,j}$ stored in on-chip RAM 1 is used as $\psi_{i,j}$. Otherwise, $\psi_{i,j}$ should be computed using Equation (1). It can be easily shown that only $\psi_{-1,j}$, $\psi_{N+1,j}$, $\psi_{i,-1}$, and $\psi_{i,N+1}$, $0 \leq i \leq N$, $0 \leq j \leq N$ actually require mirror reflection for the computation of $\gamma_{0,j}$, $\gamma_{N,j}$, $\gamma_{i,0}$ and $\gamma_{i,N}$. Using Equation (1), it follows that $\psi_{-1,j} = \zeta_{1,j}$, $\psi_{N+1,j} = \zeta_{N-1,j}$, $\psi_{i,-1} = \zeta_{i,1}$, and $\psi_{i,N+1} = \zeta_{i,N-1}$, it is not necessary to design a circuit for mirror reflection for the pre-transform unit. We only have to reconfigure the address generator in the unit so that when $\psi_{-1,j}$, $\psi_{N+1,j}$, $\psi_{i,-1}$, or $\psi_{i,N+1}$ are desired, the address of $\zeta_{1,j}$, $\zeta_{N-1,j}$, $\zeta_{i,1}$ or $\zeta_{i,N-1}$ will be delivered to on-chip RAM 1, respectively.

Figure 3. **(a)** The architecture of phase-wrapping unit in the pre-computation unit, **(b)** The architecture of each module K in the phase-wrapping unit.

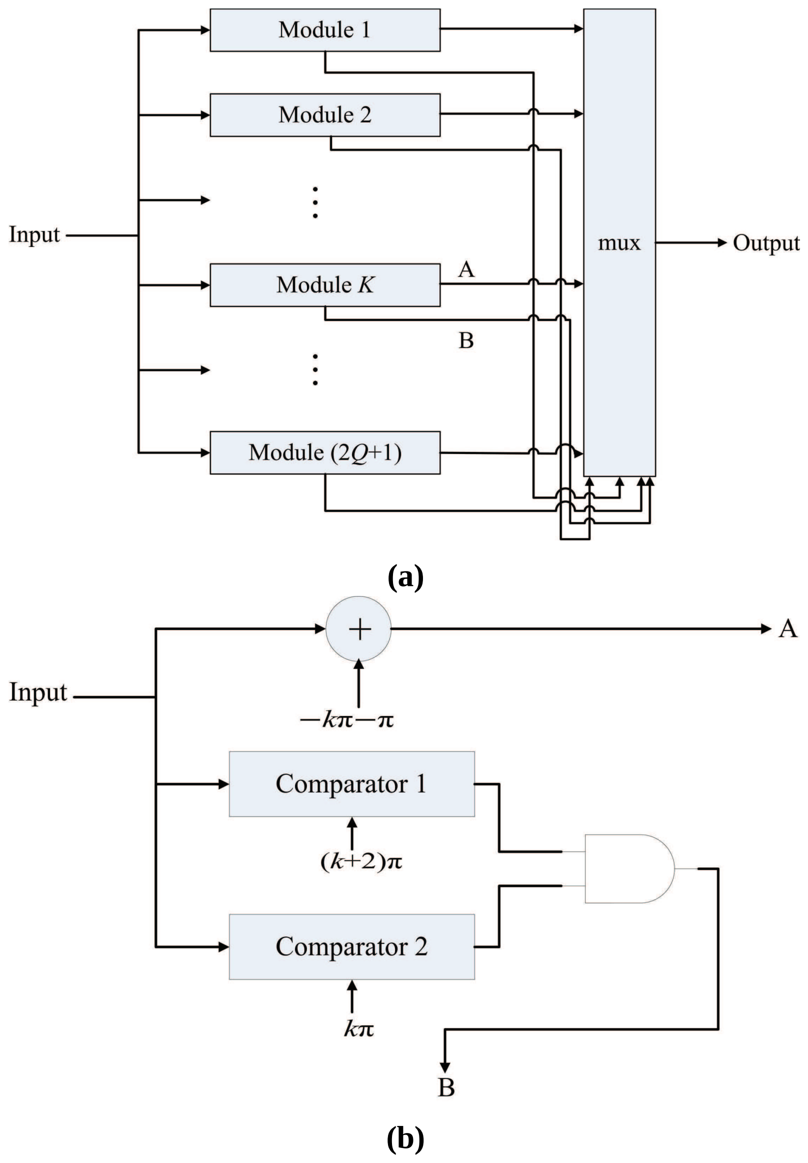


Figure 4. The timing diagram for pipeline operation at the first step of pre-transform unit.

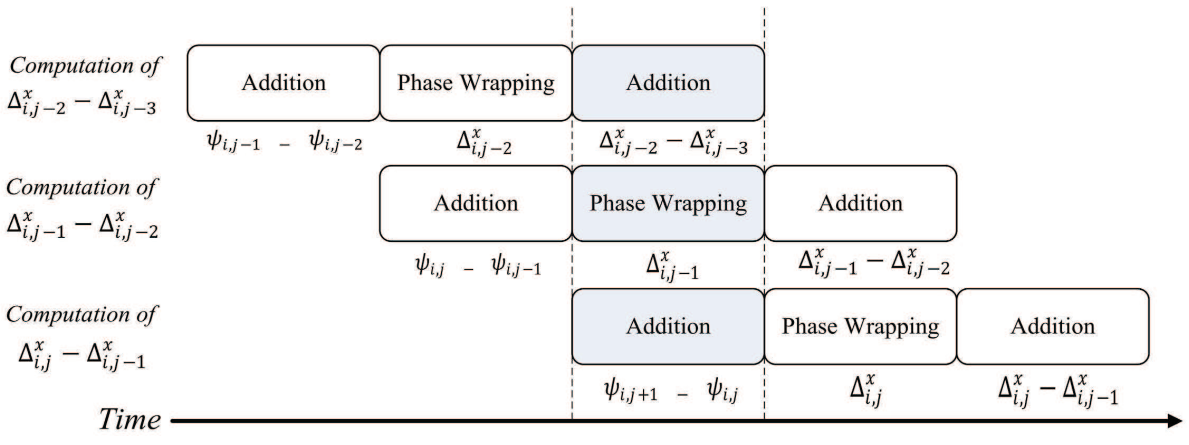
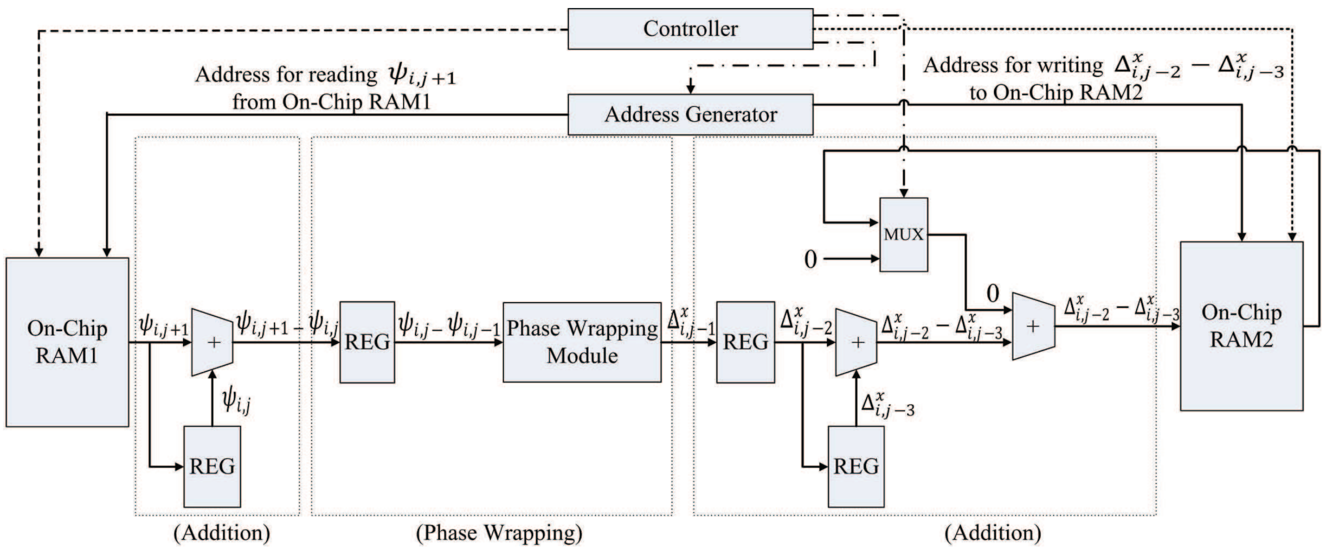
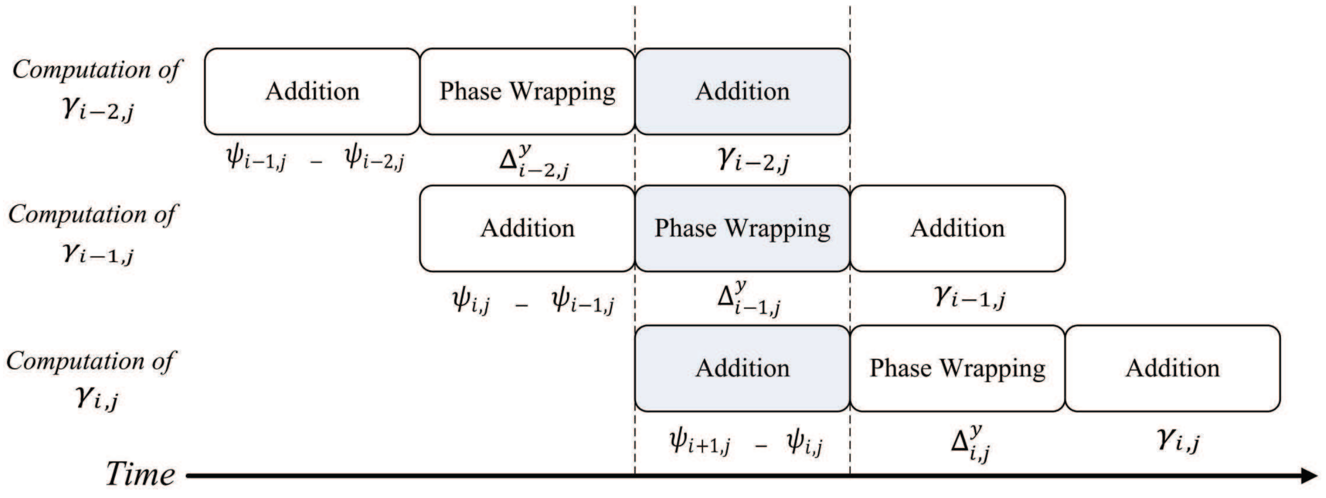

Computation of $\Delta^x_{i,j-2} - \Delta^x_{i,j-3}$
Addition
Phase Wrapping
Addition
$\psi_{i,j-1}$ − $\psi_{i,j-2}$
$\Delta^x_{i,j-2}$
$\Delta^x_{i,j-2} - \Delta^x_{i,j-3}$
Computation of $\Delta^x_{i,j-1} - \Delta^x_{i,j-2}$
Addition
Phase Wrapping
Addition
$\psi_{i,j}$ − $\psi_{i,j-1}$
$\Delta^x_{i,j-1}$
$\Delta^x_{i,j-1} - \Delta^x_{i,j-2}$
Computation of $\Delta^x_{i,j} - \Delta^x_{i,j-1}$
Addition
Phase Wrapping
Addition
$\psi_{i,j+1}$ − $\psi_{i,j}$
$\Delta^x_{i,j}$
$\Delta^x_{i,j} - \Delta^x_{i,j-1}$
Time

Figure 5. The input/output to each stage of the pipeline for the shaded time interval marked in the Figure 4.



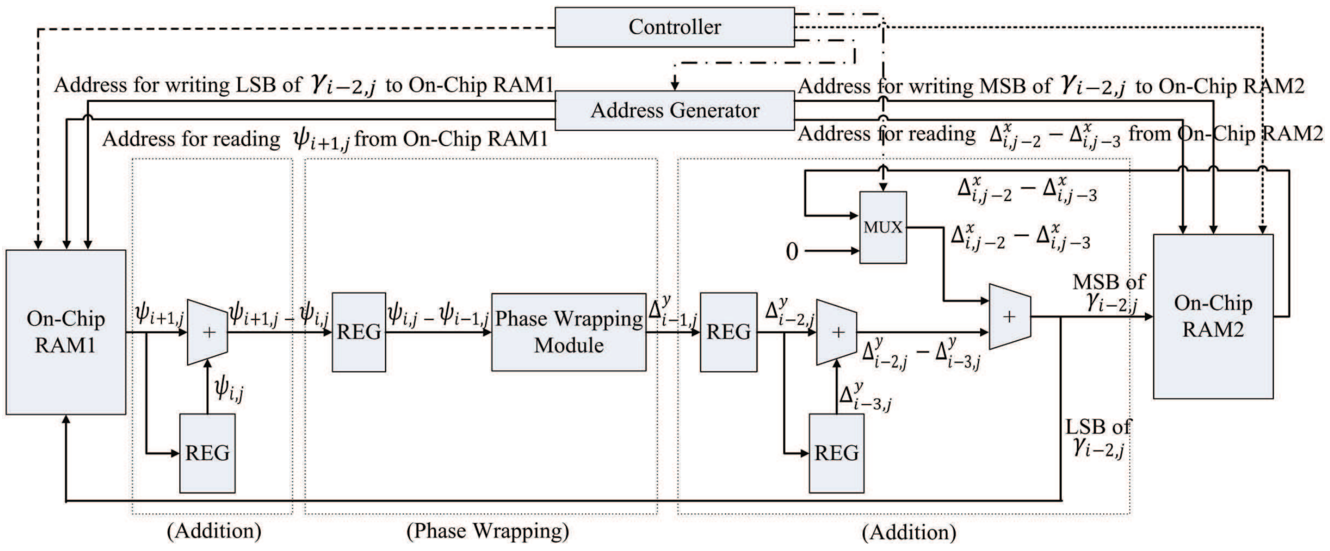
Controller
Address for reading $\psi_{i,j+1}$ from On-Chip RAM1
Address Generator
Address for writing $\Delta^x_{i,j-2} - \Delta^x_{i,j-3}$ to On-Chip RAM2
On-Chip RAM1
$\psi_{i,j+1}$
+
$\psi_{i,j+1} - \psi_{i,j}$
$\psi_{i,j}$
REG
REG
$\psi_{i,j} - \psi_{i,j-1}$
Phase Wrapping Module
$\Delta^x_{i,j-1}$
REG
$\Delta^x_{i,j-2}$
+
$\Delta^x_{i,j-2} - \Delta^x_{i,j-3}$
$\Delta^x_{i,j-3}$
REG
MUX
0
0
+
$\Delta^x_{i,j-2} - \Delta^x_{i,j-3}$
On-Chip RAM2
(Addition)
(Phase Wrapping)
(Addition)

Figure 6. The timing diagram for pipeline operation at the second step of pre-transform unit.



Another advantage of the employment of address generator is that it is able to generate multiple addresses for the concurrent read and write accesses of the on-chip memory. Multiple address generation is essential for the implementation of the pipeline in the pre-transform unit. For the shaded time interval indicated in Figure 7, the retrieval of $\psi_{i+1,j}$ and $\Delta^x_{i,j-2} - \Delta^x_{i,j-3}$ are required at stages 1 and 3, respectively. In addition, the computation result at stage 3, $\gamma_{i-2,j}$, should also be written to the RAM 1 and RAM 2. As shown in Figure 7, the address generator sends three different addresses to the on-chip memory for the concurrent access: address for reading $\psi_{i+1,j}$ from RAM 1, address for reading $\Delta^x_{i,j-2} - \Delta^x_{i,j-3}$ from RAM 2, and address for writing $\gamma_{i-2,j}$ to RAM 1 and RAM 2. Other alternatives for memory accesses are based on CPU or direct memory access (DMA). However, because there is only one memory access at a time, using the CPU or DMA-based memory accesses for the proposed pipeline architecture may be difficult.

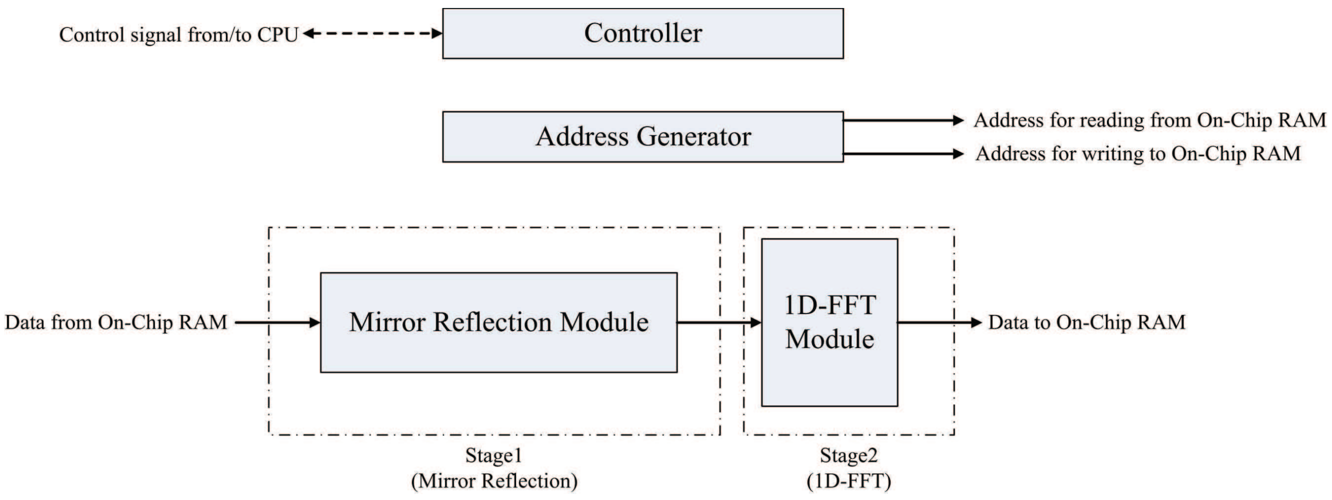
Figure 7. The input/output to each stage of the pipeline for the shaded time interval marked in the Figure 6.


Controller
Address Generator
Address for writing LSB of $\gamma_{i-2,j}$ to On-Chip RAM1
Address for reading $\psi_{i+1,j}$ from On-Chip RAM1
Address for writing MSB of $\gamma_{i-2,j}$ to On-Chip RAM2
Address for reading $\Delta^x_{i,j-2} - \Delta^x_{i,j-3}$ from On-Chip RAM2
On-Chip RAM1
On-Chip RAM2
$\psi_{i+1,j}$
$\psi_{i+1,j} - \psi_{i,j}$
$\psi_{i,j}$
REG
$\psi_{i,j} - \psi_{i-1,j}$
Phase Wrapping Module
$\Delta^y_{i-1,j}$
$\Delta^y_{i-2,j}$
$\Delta^y_{i-2,j} - \Delta^y_{i-3,j}$
$\Delta^y_{i-3,j}$
MUX
0
$\Delta^x_{i,j-2} - \Delta^x_{i,j-3}$
MSB of $\gamma_{i-2,j}$
LSB of $\gamma_{i-2,j}$
(Addition)
(Phase Wrapping)
(Addition)

3.3. FFT Unit

The FFT unit is employed for implementing Steps 2 and 4 of the algorithm. Figure 8 shows the architecture of the FFT unit, which contains controller, address generator, mirror reflection module and one-dimensional FFT (1D-FFT) module. The FFT unit reads the source data from both on-chip RAM 1 and RAM 2. The results produced by the FFT unit are then stored back to on-chip RAM 1 and RAM 2 to replace the source data.

Figure 8. The architecture of FFT unit.


Control signal from/to CPU
Controller
Address Generator
Address for reading from On-Chip RAM
Address for writing to On-Chip RAM
Data from On-Chip RAM
Mirror Reflection Module
1D-FFT Module
Data to On-Chip RAM
Stage1 (Mirror Reflection)
Stage2 (1D-FFT)

In the FFT unit, each row of $\gamma_{i,j}$ is loaded from the on-chip memory one at a time. The FFT unit then writes the computational results directly back to the same row in the on-chip memory. After the row operations are completed, the column operations will proceed in the same manner. After the completion of all the column operations, the array stored in the on-chip RAM is $\Gamma_{m,n}$, the two-dimensional FFT of $\gamma_{i,j}$.

From Step 2.1 of the phase unwrapping algorithm, it follows that the mirror reflection is required before the 1D-FFT transform (or inverse transform). The mirror reflection module is a $2N$-stage shift register. The input to the shift register is located at the stage $N + 1$. The first $N + 1$ stages of the shift register are able to operate in two directions for the implementation of mirror reflection. The output of the shift register is connected to the 1D-FFT module. The proposed module operates in two phases using the shift register. At the first phase, each data point entering the mirror reflection module is shifted in two directions. After all the data points in a row have entered the shift register, all the data points are shifted right one at a time to the output at the second phase.

We use Altera FFT MegaCore function [26] to implement the 1D-FFT module. The transform length of the FFT is $2N$. The 1D-FFT module has single data input and single data output. The input/output dataflow of the module operates in streaming mode, allowing the continuous process of input data stream, as well as producing the continuous output data stream. In addition, the module contains only one butterfly processor. Therefore, it requires only a single complex multiplier for the FFT implementation [26]. The area cost can then be minimized.

The FFT unit is able to operate as a two-stage pipeline, where the first stage is mirror reflection, and the second stage is 1D-FFT. Figure 9 shows the timing diagram for the pipeline operation of the FFT unit for the rows of the array $\gamma_{i,j}$. Note that the operation of each stage of FFT unit is separated into two phases. Both the stages will operate at the same phase at the same time. Figures 10 and 11 show the operation of phase 1 and phase 2 at each stage, respectively. As shown in Figure 10, at the phase 1 of the mirror reflection, a row of $\gamma_{i,j}$ (e.g., i-th row) is loaded from on-chip RAM for computing $\lambda_{i,j}$, $0 \le j < 2N$. At the same time, the phase 1 of 1D-FFT module uses $\lambda_{i-1,j}$, $0 \le j < 2N$ as the input for computing $\Lambda_{i-1,n}$, $0 \le n < 2N$. The mirror reflection module then delivers $\lambda_{i,j}$, $0 \le j < 2N$ to the 1D-FFT module at its phase 2 operation. At the same time, the 1D-FFT module sends the $\Lambda_{i-1,n}$, $0 \le n < N$, to the on-chip RAM, as depicted in Figure 11.

Note that the FFT unit is also used for the computation of inverse 2D-FFT of $\Phi_{m,n}$. The data stored in the on-chip memory is $\Phi_{m,n}$. The 1D-FFT module will operate as the 1D inverse FFT for the input data. The FFT unit will then produce $\bar{\phi}_{i,j}$ to the on-chip memory.

Figure 9. The timing diagram for pipeline operation of FFT unit.

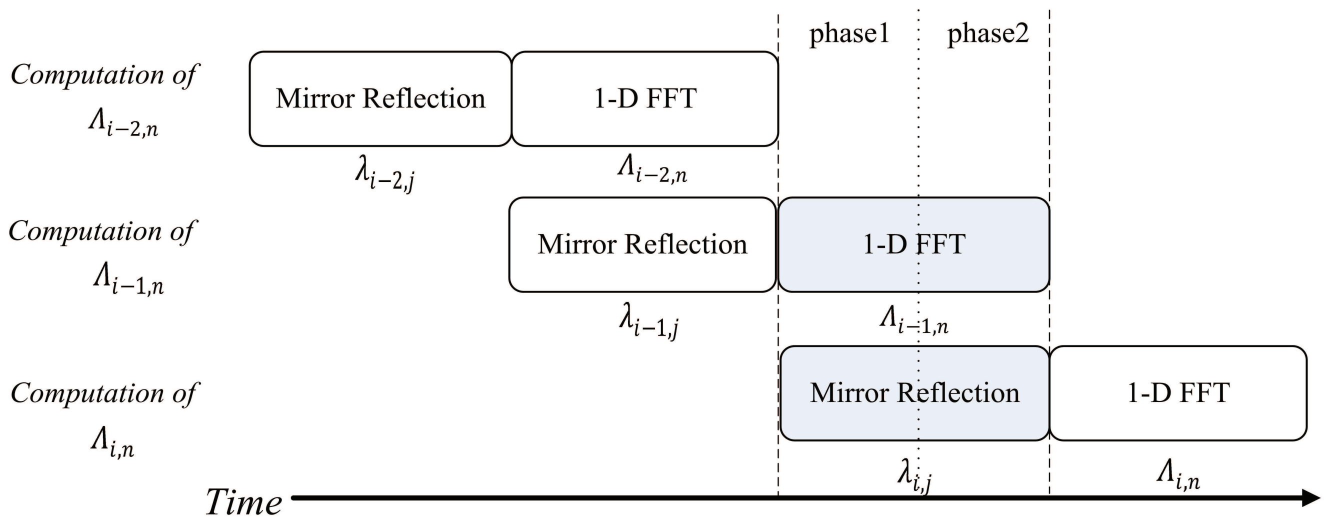


Figure 10. The operation of phase 1 at each stage of the pipeline for the shaded time interval marked in the Figure 9.

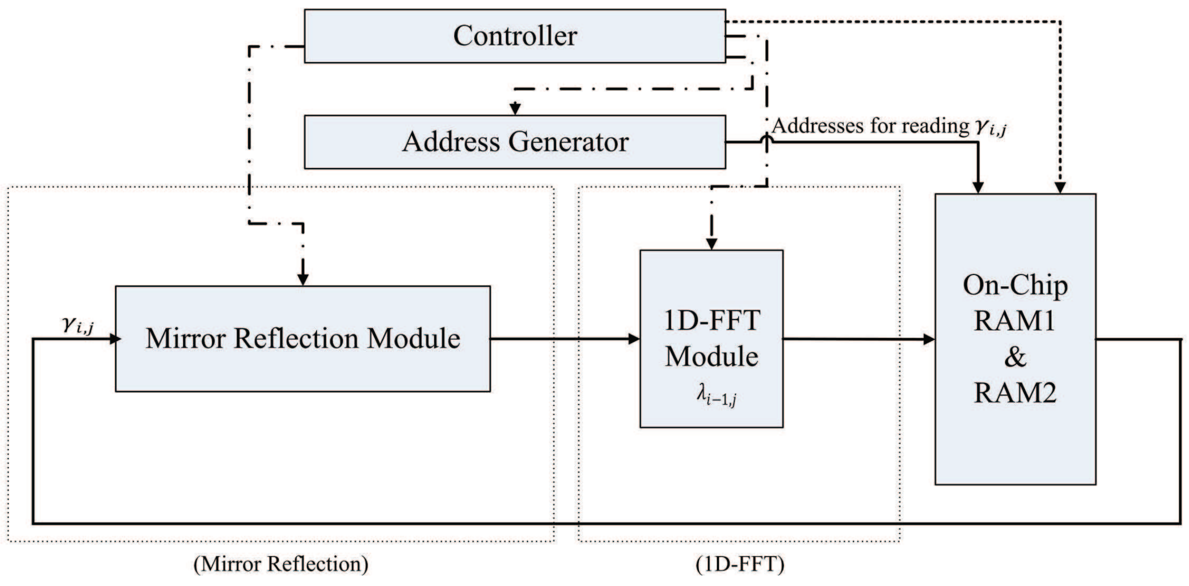
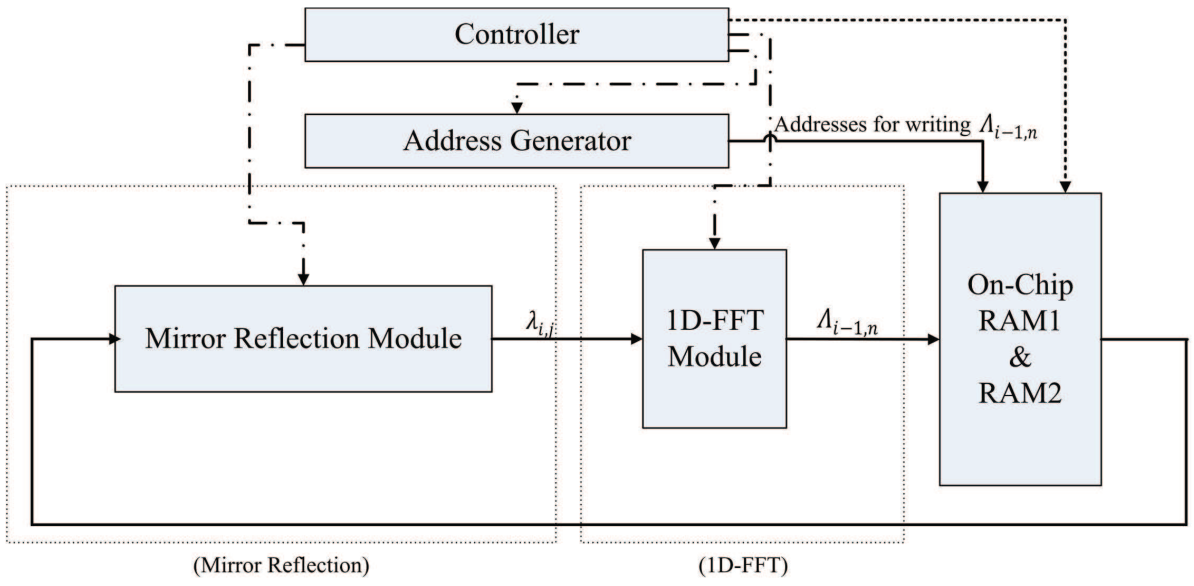


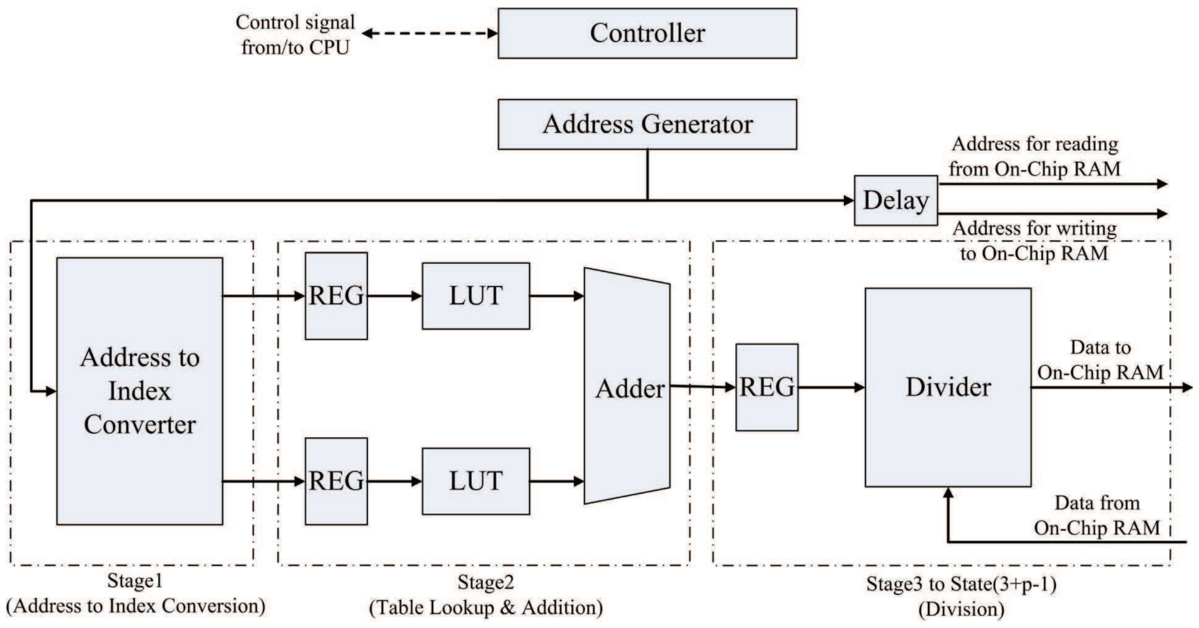
Figure 11. The operation of phase 2 at each stage of the pipeline for the shaded time interval marked in the Figure 9.



3.4. Post-Transform Unit

The post-transform unit is used for the hardware computation of Step 3 in the algorithm. Therefore, the objective of the post-transform unit is to realize Equation (6) in hardware. Figure 12 shows the architecture of the post-transform unit, which consists of two cosine computation modules, a divider, and adders. The goal of the two cosine computation modules is to compute $\cos(\pi m/N), m = 0, ..., N-1$, and $\cos(\pi n/N), n = 0, ..., N-1$, respectively. Since m and n only takes N possible values for the cosine transform, the two modules can be implemented as a simple look-up table (LUT), consisting of N entries. The i-th entry of the table in the two cosine computation modules contains the value of $\cos(\pi i/N)$.

Figure 12. The architecture of post-transform unit.


Control signal from/to CPU
Controller
Address Generator
Delay
Address for reading from On-Chip RAM
Address for writing to On-Chip RAM
Address to Index Converter
REG
LUT
REG
LUT
Adder
REG
Divider
Data to On-Chip RAM
Data from On-Chip RAM
Stage1 (Address to Index Conversion)
Stage2 (Table Lookup & Addition)
Stage3 to State(3+p-1) (Division)

Although LUTs can be used for the implementation of cosine modules, they may be difficult to be used for the design of divider in the post-transform unit. From Equation (6), we can see that the nominator and denominator of the divider are all real numbers. Consequently, the number of entries of the LUT will be very high when it is used for divider design. A general divider which accepts real numbers as the inputs is then desired. The divider in the architecture is implemented by Altera Floating Point Megafunction ALTFP_DIV [27]. The divider is separated into p pipeline stages to enhance the throughput of the post-transform unit.

Given $\Gamma_{m,n}$ in the on-chip memory, the post-transform unit operates as follows. The unit loads the FFT coefficients $\Gamma_{m,n}$ from the on-chip memory one at a time based on the raster scan order. To reduce the amount of bus traffic, the address delivered to the on-chip memory for loading $\Gamma_{m,n}$ is also delivered to the post-transform unit for extracting the indices m and n, which are then delivered to the cosine computation modules for finding $\cos(\pi m/N)$ and $\cos(\pi n/N)$. Both $\Gamma_{m,n}$ loaded from the on-chip memory and $(2\cos(\pi m/N) + 2\cos(\pi n/N) - 4)$ computed by the adders are then used as the input to the divider for computing $\Phi_{m,n}$. The output of the divider is then stored directly back to the on-chip memory.

The post-transform unit is implemented as a $(2 + p)$-stage pipeline. As shown in Figure 12, the first stage performs the address to index conversion. That is, the address used for retrieving $\Gamma_{m,n}$ from the on-chip RAM is used for computing indices m and n at this stage. The second stage of the pipeline computes $\cos(\pi m/N)$ and $\cos(\pi n/N)$ based on table look-up method. In addition, the second stage consists of an adder for computing $(2\cos(\pi m/N) + 2\cos(\pi n/N) - 4)$. The third stage to the $(2 + p)$-th stage of the pipeline form a p-stage divider for computing $\Phi_{m,n}$. Figure 15 shows the timing diagram for the pipeline operation of the post-transform unit. The input/output to each stage of the pipeline for the shaded time interval marked in the Figure 13 is then depicted in Figure 14.

The major advantage of the design is that only the $\Gamma_{m,n}$ is required from the input ports. The terms $\cos(\pi m/N)$ and $\cos(\pi n/N)$ can be obtained from the address for retrieving $\Gamma_{m,n}$ from the on-chip

RAM. Based on the address, the computation of $\cos(\pi m/N)$ and $\cos(\pi n/N)$ are then carried out by simple LUT. Consequently, the single address for retrieving $\Gamma_{m,n}$ actually produces $\Gamma_{m,n}$, $\cos(\pi m/N)$ and $\cos(\pi n/N)$ for computing $\Phi_{m,n}$ using Equation (6). This single-address-multiple-data scheme is beneficial for enhancing the computational speed of the post-transform unit.

Figure 13. The timing diagram for pipeline operation at post-transform unit for p = 2.

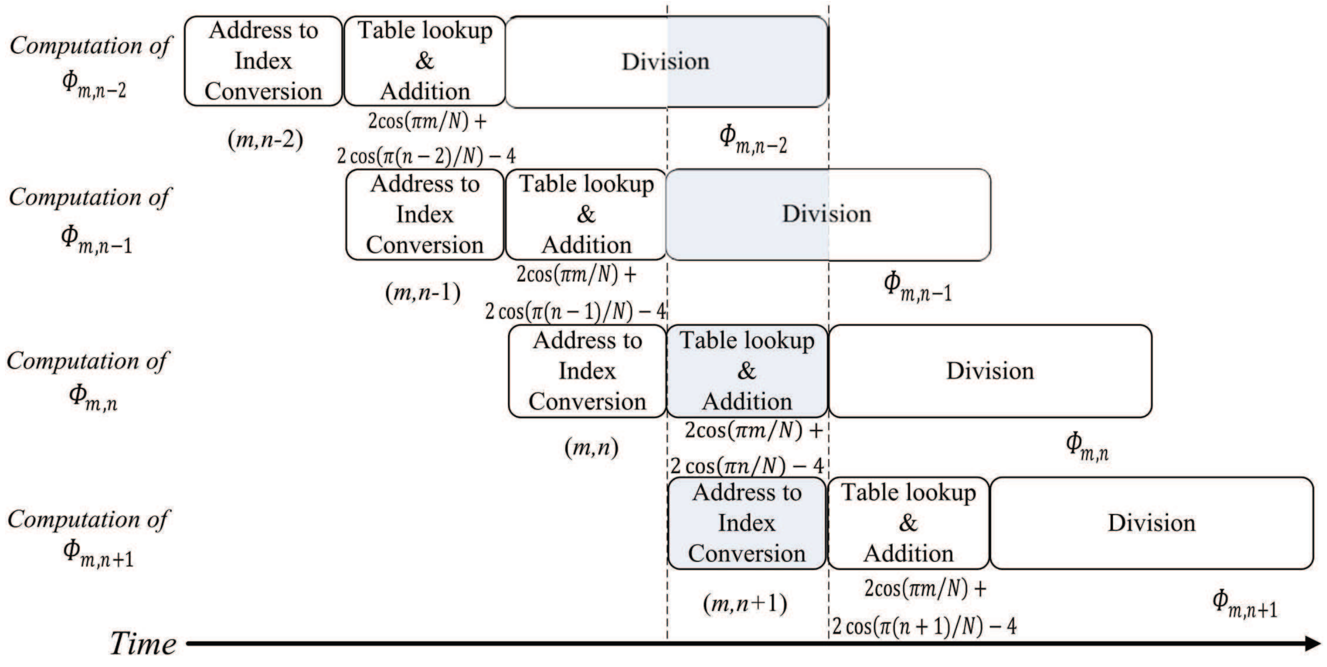
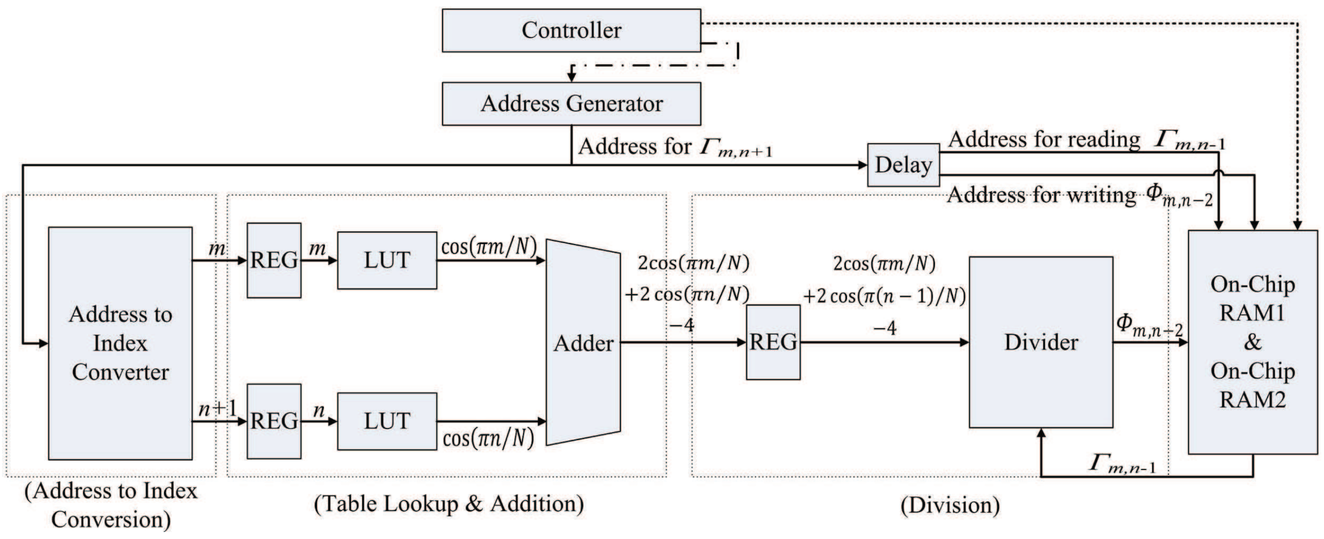

Computation of $\Phi_{m,n-2}$
Address to Index Conversion
Table lookup & Addition
Division
$(m,n\text{-}2)$
$2\cos(\pi m/N) + 2\cos(\pi(n-2)/N) - 4$
$\Phi_{m,n-2}$
Computation of $\Phi_{m,n-1}$
Address to Index Conversion
Table lookup & Addition
Division
$(m,n\text{-}1)$
$2\cos(\pi m/N) + 2\cos(\pi(n-1)/N) - 4$
$\Phi_{m,n-1}$
Computation of $\Phi_{m,n}$
Address to Index Conversion
Table lookup & Addition
Division
(m,n)
$2\cos(\pi m/N) + 2\cos(\pi n/N) - 4$
$\Phi_{m,n}$
Computation of $\Phi_{m,n+1}$
Address to Index Conversion
Table lookup & Addition
Division
$(m,n+1)$
$2\cos(\pi m/N) + 2\cos(\pi(n+1)/N) - 4$
$\Phi_{m,n+1}$
Time

Figure 14. The input/output to each stage of the pipeline for the shaded time interval marked in the Figure 13.


Controller
Address Generator
Address for $\Gamma_{m,n+1}$
Delay
Address for reading $\Gamma_{m,n-1}$
Address for writing $\Phi_{m,n-2}$
Address to Index Converter
m
$n+1$
REG
REG
m
n
LUT
LUT
$\cos(\pi m/N)$
$\cos(\pi n/N)$
Adder
$2\cos(\pi m/N) +2\cos(\pi n/N) -4$
REG
$2\cos(\pi m/N) +2\cos(\pi(n-1)/N) -4$
Divider
$\Phi_{m,n-2}$
$\Gamma_{m,n-1}$
On-Chip RAM1 & On-Chip RAM2
(Address to Index Conversion)
(Table Lookup & Addition)
(Division)

3.5. *Analysis of Area Costs and Speed*

Two types of performance are considered in this paper: the latency and area complexities. The latency of each unit is defined as the time required for finishing the operations of that unit. Because the arithmetic operators and storage cells are the basic building blocks for the architecture, the area complexities are separated into 2 categories: the number of arithmetic operators, and the number of storage cells. The arithmetic operators consist of adders, multipliers, and dividers. The storage cells contain registers, ROM cells and RAM cells.

Table 1 summarizes the area complexities and latency of the proposed architecture. Recall that the images considered in the proposed architecture are of image resolution (N+1) × (N+1). It can then be observed from Table 1 that the number of arithmetic operators in the proposed architecture is independent of the image resolution. In addition, the number of storage cells grows linearly with the image resolution.

Table 1. Area complexities and latency of the proposed architecture with respect to the image resolution (N+1) × (N+1), where the function O, termed big O function, is used to indicate the asymptotic complexities of the architecture.

	Pre-Transform	FFT	Post-Transform	On-Chip Memory	Overall
Arithmetic Operators	$O(1)$	$O(1)$	$O(1)$	0	$O(1)$
Storage Cells	$O(1)$	$O(N)$	$O(N)$	$O(N^2)$	$O(N^2)$
Latency	$O(N^2)$	$O(N^2 \log N)$	$O(N^2)$		$O(N^2 \log N)$

The number of arithmetic operators is independent of the image resolution because each unit in the proposed architecture only uses a fixed number of adders, multipliers and/or dividers, independent of N. For example, the FFT unit only employs one complex multiplier in the 1D FFT module. The post-transform unit also adopts only one divider.

The number of storage cells used by the proposed architecture increases with the image resolution. For the FFT unit, because the mirror reflection module contains $2N$ registers, its complexity is $O(N)$. For the post-transform unit, the size of tables for cosine transform is also proportional to N. The number of storage cells in the on-chip RAM1 and RAM2 is (N+1) × (N+1). Its complexity therefore is $O(N^2)$, and grows linearly with the image resolution.

To evaluate the time complexity, we first note that the pre-transform and post-transform units need to perform additions and division to each of the $(N + 1) \times (N + 1)$ input data points, respectively. Since the number of adders and dividers are independent of N, the latency of these units is $O(N^2)$. For the 2D FFT and 2D IFFT operations, the latency is given by $O(N^2 logN)$.

3.6. Software-Hardware Co-Design

The proposed architecture is used as a custom user logic in a SOPC system consisting of softcore NIOS CPU [28], and the proposed architecture, as depicted in Figure 15.

The objective of NIOS CPU is to control the data flow of the proposed architecture. Note that the on-chip RAM in the proposed architecture provides the source data for pre-transform unit, FFT unit and post-transform unit. The on-chip RAM also stores the computation results from these units. To ensure that the data in on-chip RAM is delivered to the correct unit, and the computation results of each unit can be sent to the on-chip RAM, the CPU is responsible for activating controller at each unit, and specifying the proper value in the status register in the on-chip RAM, which controls the multiplexer in the read and write ports of the memory.

Figure 15. The SOPC system for phase unwrapping.

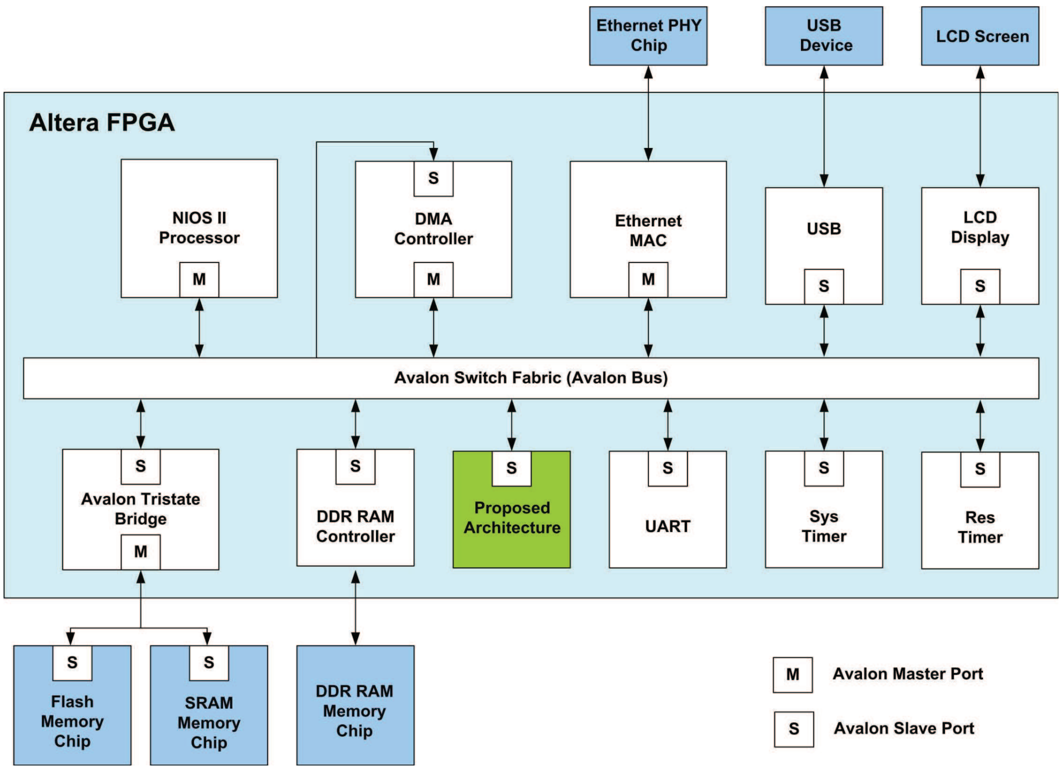
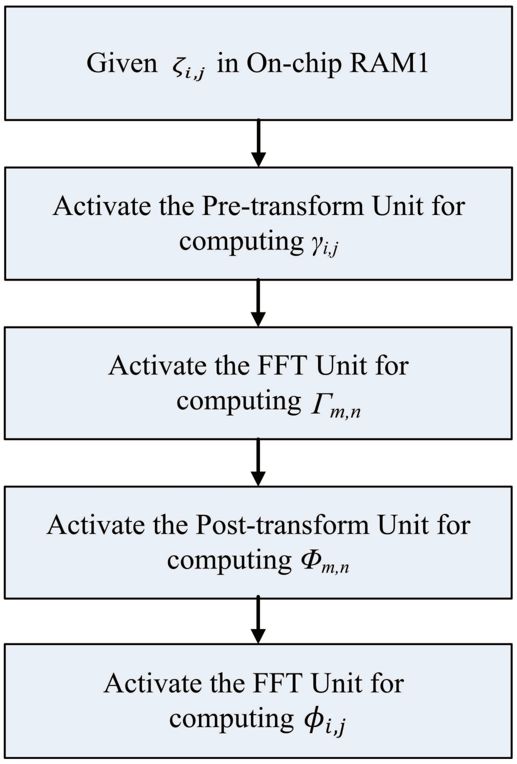


Figure 16. The flowchart of the software executing by the CPU.


Given $\zeta_{i,j}$ in On-chip RAM1
Activate the Pre-transform Unit for computing $\gamma_{i,j}$
Activate the FFT Unit for computing $\Gamma_{m,n}$
Activate the Post-transform Unit for computing $\Phi_{m,n}$
Activate the FFT Unit for computing $\phi_{i,j}$

As shown in Figure 15, the proposed architecture is connected to the Avalon bus for the communication with the NIOS CPU. A special interface circuit is designed for the connection. The interface circuit contains registers, address decoders and multiplexers so that the on-chip memory and the status registers in the controllers of all the units in the proposed architecture can be accessed by the NIOS CPU.

Figure 16 shows the flowchart of the software executed by the CPU. It can be observed from Figure 16 that the software only involves the controller activation, as well as specifying the contents of status register in the on-chip RAM. Because of the simplicity of the software, the execution of the phase-unwrapping algorithm can be significantly enhanced.

4. Experimental Results

This section presents some experimental results of the proposed architecture. The design platform is Altera Quartus II with SOPC Builder and NIOS II IDE. The target FPGA device for the hardware design is Altera Stratix III EP3SL150.

The hardware resource utilization of each unit in the proposed architecture are revealed in Tables 2–4. The images resolutions considered in these tables are 129 $\times$ 129 (*i.e.*, N $=$ 128), 257 $\times$ 257 (*i.e.*, N $=$ 256) and 513 $\times$ 513 (*i.e.*, N $=$ 512). There are three types of area costs considered in the experiment: adaptive logic modules (ALMs), embedded memory bits, digital signal processing (DSP) blocks. The ALMs are used for the implementations of arithmetic operators and storage cells. The embedded memory bits are used mainly for storage cells. The DSP blocks are used only for arithmetic operators such as dividers and multipliers. The target FPGA device Altera Stratix III EP3SL150 contains 56,800 ALMs, 5,630,976 embedded memory bits, and 384 DSP blocks.

It can be observed from Tables 2–4 that both the pre-transform and post-transform unit utilize only a small percentage of ALMs, embedded memory bits, and DSP blocks. The ALM utilization of post-transform unit grows with image resolution because the tables in the cosine computation modules are implemented by ALMs. The divider in the post-transform module is implemented by DSP blocks. The major hardware resource utilized by on-chip memory is the embedded memory bits, which grows linearly with N^2. The FFT unit utilizes most of the ALMs, which are used for the design of mirror reflection module and 1D FFT module. The FFT unit uses DSP block only for the implementation of complex multipliers. Since there is only one complex multiplier in the 1D FFT module, independent of image resolutions. The DSP block utilization of FFT unit is therefore also independent of image resolutions.

Table 2. The ALM utilization of each unit in the proposed architecture for various image resolutions.

Image Resolutions	Pre-Transform	FFT	Post-Transform	On-Chip Memory
129 $\times$ 129	197	8,358	889	136
257 $\times$ 257	201	10,532	959	178
513 $\times$ 513	221	19,049	1,641	301

Table 3. The embedded memory bit utilization of each unit in the proposed architecture for various image resolutions.

Image Resolutions	Pre-Transform	FFT	Post-Transform	On-Chip Memory
129 × 129	0	61,440	4,608	299,538
257 × 257	0	122,880	4,608	1,188,882
513 × 513	0	233,472	4,608	4,737,042

Table 4. The DSP block utilization of each unit in the proposed architecture for various image resolutions.

Image Resolutions	Pre-Transform	FFT	Post-Transform	On-Chip Memory
129 × 129	0	24	16	0
257 × 257	0	24	16	0
513 × 513	0	24	16	0

Table 5 shows the total hardware resource utilization of the proposed architecture for images with different resolutions. It can be observed from the table that the increase in the ALM utilization is small when image resolution enlarges. In fact, as the image resolution increases 16 folds (*i.e.*, from 129 × 129 to 513 × 513), the ALM utilization is only increased approximately 2 folds (*i.e.*, from 9,580 to 2,1212). The embedded memory utilization grows linearly with the image resolution. The DSP block utilization is independent of the image resolution. In addition to the area complexities of the entire architecture, the table consists of the area complexities of the entire SOPC using the proposed architecture as a custom user logic.

Table 5. The total area costs of the proposed architecture for various image resolutions.

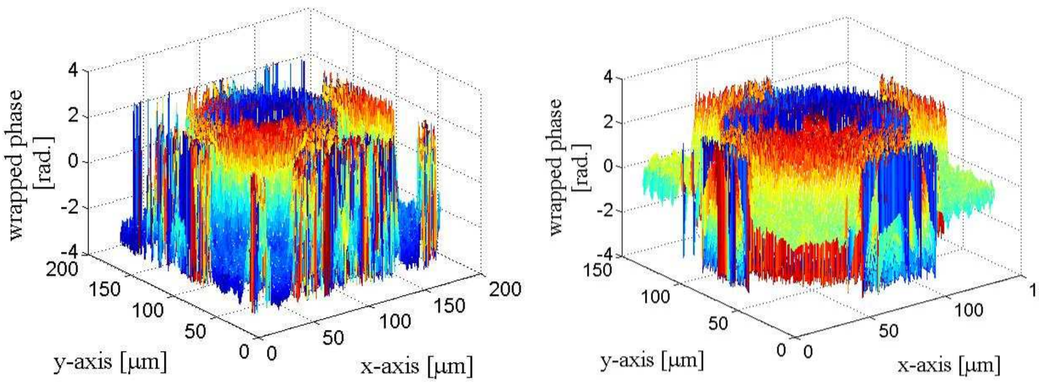
	Proposed Arch.			Entire SOPC		
Image Resolutions	ALMs	Embedded Memory Bits	DSP Blocks	ALMs	Embedded Memory Bits	DSP Blocks
129 × 129	9,580/56,800 (17%)	365,586/5,630,976 (6%)	40/384 (10%)	17,081/56,800 (30%)	968,722/5,630,976 (17%)	44/384 (11%)
257 × 257	11,870/56,800 (21%)	1,316,370/5,630,976 (23%)	40/384 (10%)	20,905/56,800 (36%)	1,916,434/5,630,976 (34%)	44/384 (11%)
513 × 513	21,212/56,800 (37%)	4,975,122/5,630,976 (88%)	40/384 (10%)	28,568/56,800 (50%)	5,085,778/5,630,976 (90%)	44/384 (11%)

The execution time of each step of phase unwrapping algorithm implemented by the proposed architecture for various image resolutions is shown in Table 6. The execution time is measured by the SOPC platform with 500 MHz NIOS softcore CPU. The proposed architecture is used as the accelerator, as shown in Figure 17. It can be observed from the table that the FFT and IFFT are the most time consuming operations. In particular, when image resolution is 513 × 513, both the FFT and IFFT consume 71.9 % of the total computational time. These results are consistent with those shown in Table 1, where the FFT unit has the highest time complexity.

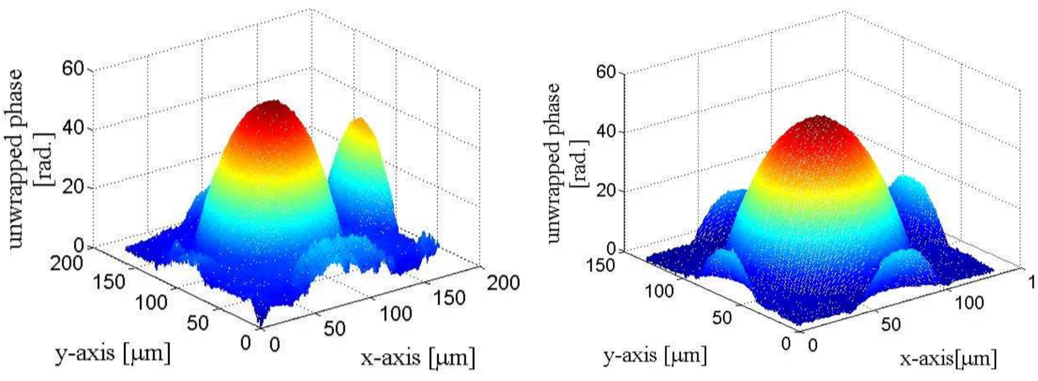
Table 6. The execution time of the proposed phase unwrapping architecture for various image resolutions.

Image Resolutions	Pre-Transform	FFT	Post-Transform	Inverse FFT	Total
129 × 129	0.1 (ms)	0.3 (ms)	0.2 (ms)	0.3 (ms)	0.9 (ms)
257 × 257	0.3 (ms)	0.9 (ms)	0.4 (ms)	0.9 (ms)	2.5 (ms)
513 × 513	1.1 (ms)	3.2 (ms)	1.4 (ms)	3.2 (ms)	8.9 (ms)

Figure 17. The phase unwrapping results for 257×257 images: **(a)** The phase wrapped image, **(b)** The phase unwrapped image produced by the proposed architecture.



(a)



(b)

Although the division and cosine operations are required in the post-transform unit, the computation time is low and comparable to that of the pre-transform unit. For a 513 × 513 image, the post-transform unit consumes only 15.7% of the total computation time. The fast computation is due to the efficient single-address-multiple-data operations as revealed in Figures 14 and 15. The single address for retrieving $\Gamma_{m,n}$ actually produces $\Gamma_{m,n}$, $\cos(\pi m/N)$ and $\cos(\pi n/N)$ for computing $\Phi_{m,n}$ in a pipeline fashion. As a result, the architecture is able to minimize number of memory accesses while maintaining high throughput.

Table 7 shows the speed of the proposed phase unwrapping architecture for images with various resolutions. The speed of its Matlab software counterpart running on the 2.8 GHz Intel I7 quad-core processor with 4 GB DDR III is also included in the table for comparison purpose. We can see from Table 7 that the proposed hardware circuit operates at significantly faster speed. The speedup over its software counterpart grows with the image resolutions. As the image resolution reaches 513 $\times$ 513, the speedup becomes 605. Note that the total computation time of phase unwrapping for 513 $\times$ 513 images is only 8.9 ms. This implies that the proposed architecture can support rendering with frame rate 100 fps. The proposed architecture therefore can be effectively employed for embedded DHM requiring rendering with high frame rate.

Table 7. The execution time of the proposed phase unwrapping architecture for various image resolutions.

Image Resolutions	Proposed Architecture	Software Counterpart	Speedup
129 $\times$ 129	0.9 (ms)	468 (ms)	585
257 $\times$ 257	2.5 (ms)	1504 (ms)	601
513 $\times$ 513	8.9 (ms)	5389 (ms)	605

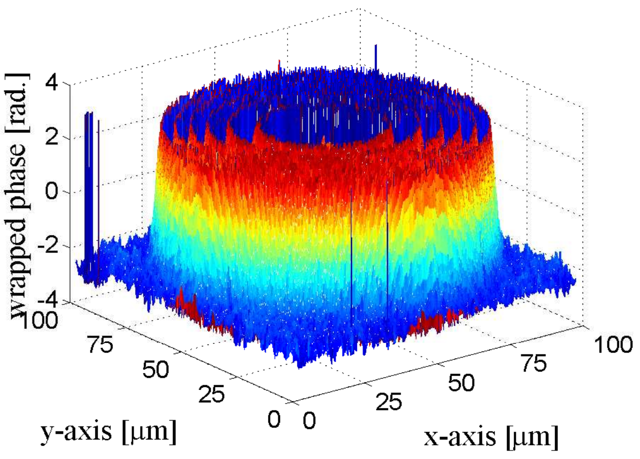
Table 8 lists the computation time of various existing phase unwrapping implementations. The direction comparisons of these implementations may be difficult because these implementations are based on different phase unwrapping algorithms with different image resolutions. In addition, these implementations are realized by different platforms. Nevertheless, we can still see from Table 8 that the proposed is an effective alternative for phase unwrapping when the computation time is an important concern.

Table 8. The execution time of different phase unwrapping implementations.

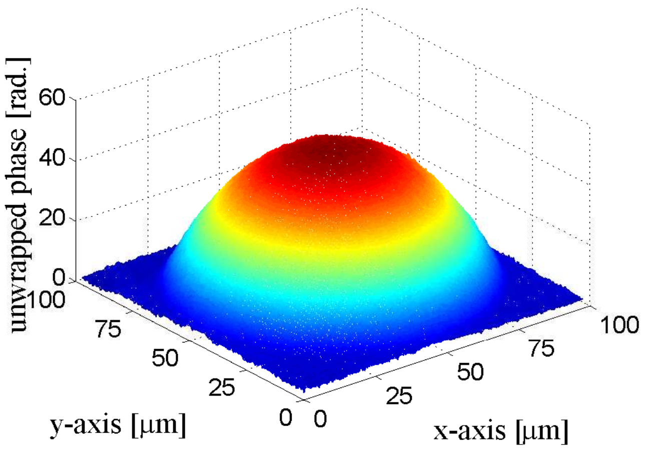
Implementations	Computation Time	Image Resolutions	Platforms
Proposed Architecture	8.9 (ms)	$513\times$ 512	FPGA (Altera Stratix III EP3SL150)
[19]	672 (ms)	$512\times$ 512	GPU (NVIDIA Geforce 8800GTX)
[21]	2.8 (s)	$640\times$ 480	GPU (NVIDIA Geforce 8800GTX)
[20]	24.7 (s)	$1,024\times$ 512	FPGA (Xilinx Vertex II Pro)

Figures 17 and 18 show the phase unwrapping results of the proposed architecture. The images considered in the experiments are produced by the DHMs in the IOP lab at the Institute of Electro-Optical Science and Technology, National Taiwan Normal University. In the experiments, microlens made by SUSSA (with radius of curvature 120 microns) are tested. The image resolution is 257 $\times$ 257 and 513 $\times$ 513 for the images shown in Figures 17 and 18, respectively. To evaluate the accuracy of the reconstruction, the radius of the curvature in the phase unwrapped results are measured, and are compared with the actual radius of the curvature of the microlens. The measured radius of curvature of the two microlens in Figure 17 are 121.0 and 121.1 microns, respectively. The measured radius of curvature of the microlen in Figure 18 is 119.7 microns. The maximum error of the unwrapped results therefore is only 1.1 microns for the measurement of radius of curvature.

Figure 18. The phase unwrapping results for 513×513 image: **(a)** The phase wrapped image, **(b)** The phase unwrapped image produced by the proposed architecture.



(a)



(b)

5. Concluding Remarks

The proposed architecture has been found to be effective for phase unwrapping. It utilizes low hardware resources. Only a single divider and complex multiplier is used in the architecture. The utilization of DSP blocks therefore is minimized. The ALM and memory bits utilization also only grow linearly with the image resolutions. Each unit in the architecture is implemented in a pipeline fashion for enhancing the throughput. The architecture therefore has fast computation speed. In particular, when the image resolution is 513×513, the computation time is only 8.1 ms. The speedup attains 605 over its software counterpart. The architecture is able to support frame rate above 100 fps for embedded

DHM rendering. The architecture is an effective alternative for the implementation of embedded DHM systems where low hardware resource utilization, high image resolution and high image rendering rate are desired.

Acknowledgements

The work was financially supported in part by National Taiwan Normal University (NTNU100-D-01).

References

1. Cuche, E.; Marquet, P.; Depeursinge, C. Simultaneous amplitude-contrast and quantitative phase-contrast microscopy by numerical reconstruction of Fresnel of-axis holograms. *Appl. Opt.* **1999**, *38*, 6994-7001.
2. Mann, C.J.; Yu, L.; Lo, C.M.; Kim, M.K. High-resolution quantitative phase-contrast microscopy by digital holography. *Opt. Express* **2005**, *13*, 8693-8698.
3. Lin, Y.C.; Cheng, C.J. Determining the refractive index profile of micro-optical elements using transflective digital holographic microscopy. *J. Opt.* **2010**, *12*, 115402.
4. Lin, Y.C.; Cheng, C.J.; Poon, T.C. Optical sectioning with a low coherence phase-shifting digital holographic microscope. *Appl. Opt.* **2011**, *50*, B25-B30.
5. Parshall, D.; Kim, M.K. Digital holographic microscopy with dual-wavelength phase unwrapping. *Appl. Opt.* **2006**, *45*, 451-459.
6. Li, Z.; Bao, Z.; Suo, Z. A joint image coregistration, phase noise suppression, and phase unwrapping method based on subspace projection for multibaseline InSAR systems. *IEEE Trans. Geosci. Remote* **2007**, *45*, 584-591.
7. Chavez, S.; Xiang, Q.S.; An, L. Understanding phase maps in MRI: A new cutline phase unwrapping method. *IEEE Trans. Med. Imaging* **2002**, *21*, 966-977.
8. Carl, D.; Fratz, M.; Pfeifer, M.; Giel, D.M.; Hofler, H. Multiwavelength digital holography with autocalibration of phase shifts and artificial wavelengths. *Appl. Opt.* **2009**, *48*, H1-H8.
9. Hansel, T.; Muller, J.; Falldorf, C.; Kopylow, C.V.; Juptner, W.; Grunwald, R.; Steinmeyer, G.; Griebner, U. Ultrashort-pulse dual-wavelength source for digital holographic two-wavelength contouring. *Appl. Phys. B* **2007**, *89*, 513-516.
10. Knoche, S.; Kemper, B.; Wernicke, G.; Bally, G.V. Modulation analysis in spatial phase shifting electronic speckle pattern interferometry and application for automated data selection on biological specimens. *Opt. Commun.* **2007**, *270*, 68-78.
11. Kuhn, J.; Colomb, T.; Montfort, F.; Charriere, F.; Emery, Y.; Cuche, E.; Marquet, P.; Depeursinge, C. Real-time dual-wavelength digital holographic microscopy with a single hologram acquisition. *Opt. Express* **2007**, *15*, 7231-7242.
12. Shaked, N.T.; Rinehart, M.T.; Wax, A. Dual-interference-channel quantitative-phase microscopy of live cell dynamics. *Opt. Lett.* **2009**, *34*, 767-769.
13. Ghiglia, D.C.; Pritt, M.D. *Two-Dimensional Phase Unwrapping: Theory, Algorithms and Software*; Wiley Inter-Science: New York, NY, USA, 1998.

14. Ghiglia, D.C.; Romero, L.A. Robust two-dimensional weighted and unweighted phase unwrapping that uses fast transforms and iterative methods. *J. Opt. Soc. Am. A* **1994**, *11*, 107-117.
15. Pritt, M.D. Phase unwrapping by means of multigrid techniques for interferometric SAR. *IEEE Trans. Geosci. Remote* **1996**, *34*, 728-738.
16. Dias, J.M.B.; Leitao, J.M.N. The ZπM algorithm: A method for interferometric image reconstruction in SAR/SAS. *IEEE Trans. Image Process.* **2002**, *11*, 408-422.
17. Dias, J.M.B.; Valadao, G. Phase unwrapping via graph cuts. *IEEE Trans. Image Process.* **2007**, *16*, 684-697.
18. Karout, S.A.; Gdeisat, M.A.; Burton, D.R.; Lalor, M.J. Two-dimensional phase unwrapping using a hybrid genetic algorithm. *Appl. Opt.* **2007**, *46*, 730-743.
19. Karasev, P.A.; Campbell, D.P.; Richards, M.A. Obtaining a 35$\times$ speedup in 2D phase unwrapping using commodity graphics processors. In *Proceedings of IEEE Radar Conference*, Boston, MA, USA, 17–20 April 2007; pp. 574-578.
20. Braganza, S.; Leeser, M. An efficient implementation of a phase unwrapping kernel on reconfigurable hardware. In *Proceedings of International Conference on Application-Specific Systems, Architectures and Processors*, Leuven, Belgium, 2–4 July 2008; pp. 138-143.
21. Mistry, P.; Braganza, S.; Kaeli, D.; Leeser, M. Accelerating phase unwrapping and affine transformations for optical quadrature microscopy using CUDA. In *Proeedings Second Workshop on General Purpose Processing on Graphics Processing Units*, Washington, DC, USA, 8–8 March 2009; pp. 28-37.
22. Shimobaba, T.; Sato, Y.; Miura, J.; Takenouchi, M.; Ito, T. Real-time digital holographic microscopy using the graphic processing unit. *Opt. Express* **2008**, *16*, 11776-11781.
23. Pritt, M.D.; Shipman, J.S. Least-squares two-dimensional phase unwrapping using FFT's. *IEEE Trans. Geosci. Remote* **1994**, *32*, 706-708.
24. Akerson, J.J.; Yang, Y.E.; Hara, Y.; Wu, B.I.; Kong, J.A. Automatic phase unwrapping algorithms in synthetic aperture radar (SAR) interferometry. *IEICE Trans. Electron.* **2000**, *E83-C*, 1896-1904.
25. Zebker, H.A.; Lu, Y. Phase unwrapping algorithms for radar interferometry: Residue-cut, least-squares, and synthesis algorithms. *J. Opt. Soc. Am. A* **1998**, *15*, 586-598.
26. *FFT MegaCore Function User Guide*; Altera Corporation: San Jose, CA, USA, 2011.
27. *Floating Point Mega Function User Guide*; Altera Corporation: San Jose, CA, USA, 2011.
28. *NIOS II Processor Reference Handbook*; Altera Corporation: San Jose, CA, USA, 2011.