Using the ac_window Classes

With Catapult Synthesis

July 2019

© 2015-2019 Mentor Graphics Corporation
All rights reserved.

This document contains information that is proprietary to Mentor Graphics Corporation. The original recipient of this document may duplicate this document in whole or in part for internal business purposes only, provided that this entire notice appears in all copies. In duplicating any part of this document, the recipient agrees to make every reasonable effort to prevent the unauthorized use and distribution of the proprietary information.

This document is for information and instruction purposes. Mentor Graphics reserves the right to make changes in specifications and other information contained in this publication without prior notice, and the reader should, in all cases, consult Mentor Graphics to determine whether any changes have been made.

The terms and conditions governing the sale and licensing of Mentor Graphics products are set forth in written agreements between Mentor Graphics and its customers. No representation or other affirmation of fact contained in this publication shall be deemed to be a warranty or give rise to any liability of Mentor Graphics whatsoever.

MENTOR GRAPHICS MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

MENTOR GRAPHICS SHALL NOT BE LIABLE FOR ANY INCIDENTAL, INDIRECT, SPECIAL, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING BUT NOT LIMITED TO LOST PROFITS) ARISING OUT OF OR RELATED TO THIS PUBLICATION OR THE INFORMATION CONTAINED IN IT, EVEN IF MENTOR GRAPHICS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

U.S. GOVERNMENT LICENSE RIGHTS: The software and documentation were developed entirely at private expense and are commercial computer software and commercial computer software documentation within the meaning of the applicable acquisition regulations. Accordingly, pursuant to FAR 48 CFR 12.212 and DFARS 48 CFR 227.7202, use, duplication and disclosure by or for the U.S. Government or a U.S. Government subcontractor is subject solely to the terms and conditions set forth in the license agreement provided with the software, except for provisions which are contrary to applicable mandatory federal laws.

TRADEMARKS: The trademarks, logos and service marks ("Marks") used herein are the property of Mentor Graphics Corporation or other parties. No one is permitted to use these Marks without the prior written consent of Mentor Graphics or the owner of the Mark, as applicable. The use herein of a third- party Mark is not an attempt to indicate Mentor Graphics as a source of a product, but is intended to indicate a product from, or associated with, a particular third party. A current list of Mentor Graphics’ trademarks may be viewed at: www.mentor.com/trademarks.

The registered trademark Linux® is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a world-wide basis.

End-User License Agreement: You can print a copy of the End-User License Agreement from: www.mentor.com/eula.

Mentor Graphics Corporation
8005 S.W. Boeckman Road, Wilsonville, Oregon 97070-7777.
Telephone: 503.685.7000
Toll-Free Telephone: 800.592.2210
Website: www.mentor.com
SupportNet: supportnet.mentor.com/

Introduction

The ac_window classes provided with Catapult were designed to allow the user to quickly implement one-dimensional and two-dimensional algorithms without having to explicitly describe the underlying memory architecture. The ac_window classes are optimized C++ libraries that have been designed to give highly optimized hardware while providing the user with a use model that is similar to the way algorithms are often generically described.

The Windowing Problem

C++ Algorithms are often written abstractly, describing only the functionality, with no consideration for the underlying memory architecture. This prevents realization of optimal hardware when synthesizing the C++. It is common for these abstract algorithms to assume that the data on which they operate is available all at once. However the hardware implementations of these algorithms usually are provided data over time. Video algorithms are a good example of this behavior where frames of video are provided to the hardware pixel-by-pixel and line-by-line. A simple one-dimensional example of this problem can be seen with the following algorithm, which accumulates samples from multiple elements of an array:

```
void accum(int x[10], int y[10]) {
  for(int I=2; I<8;I++)
    y[I] = x[I-2] + x[I-1] + x[I] + x[I+1] + x[I+2];
}
```

This algorithm describes accumulating five different samples from ***X*** as shown in Figure 1.

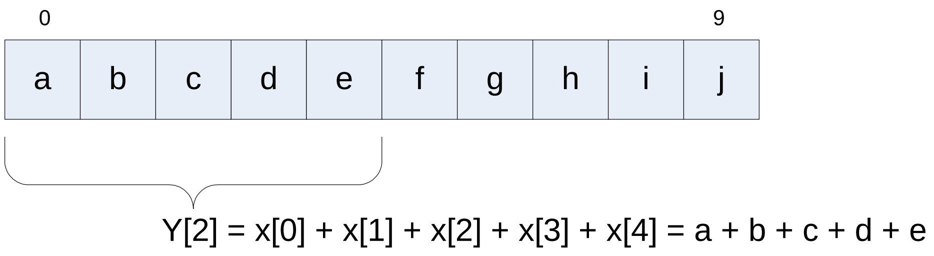

Figure 1. Array Windowing

The algorithm assumes that it has simultaneous access to all five elements of ***X***. However synthesizing this design will pose performance problems if ***X*** is only available one element at a time and the goal is to run the hardware so that one output is produced every loop iteration. In other words the main loop of this algorithm cannot be pipelined with an II=1 since that would require 5 reads of ***X*** in the same clock cycle.

This problem of implementing the hardware for the algorithm can be further complicated when boundary conditions are applied to prevent over-reading or under reading the input array. This is typically done by either clipping or mirroring the boundary array indices to keep them within the defined range of the array (Figures 2 and 3).

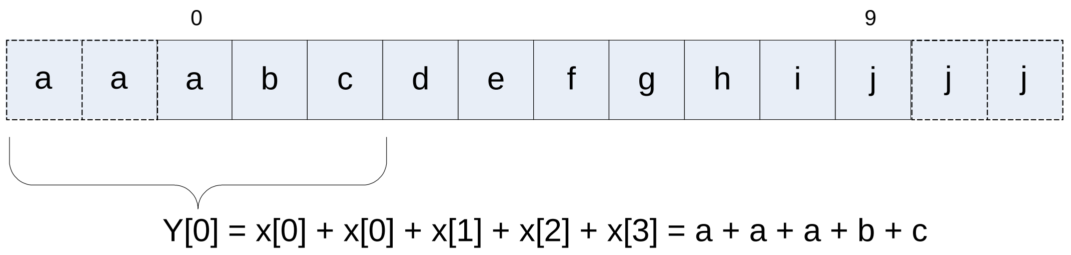

Figure 2. Effects of clipping the array index

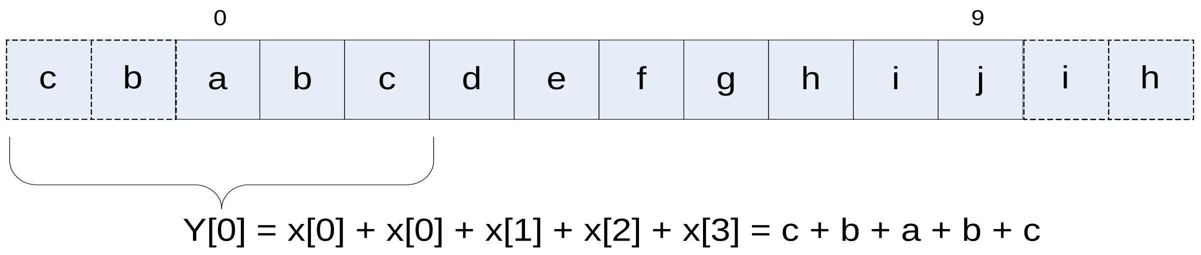

Figure 3. Effect of mirroring the array index

Two-dimensional algorithms that require similar behavior and performance as the one-dimensional case covered above are even more challenging to implement in hardware efficiently. These two-dimensional versions also often require clipping or mirroring on the array boundaries for both dimensions.

Using the ac_window classes

The ac_window class library provides several window implementations for both 1-D and 2-D algorithms. The different implementations provide the ability to re-write algorithms using several different styles. All implementations currently support odd and even size windows when using mirroring or clipping. The window classes solve the implementation bandwidth problem described above by allowing only a single array access on the design interface while “windowing” this input data into registers or memories which keeps a running history of array elements. The algorithm then operates on the stored or “windowed” data to achieve maximum performance. NOTE: All of the windowing classes expect the array data to be read sequentially and provided to the window variable in order. Thus these windowing classes would not be useful for applications that access the array data in a zigzag fashion such as motion estimation. For these types of applications a different type of windowing class is required.

Further Information

Further information is available in the Modeling Section of the *Catapult Synthesis User's and Reference Manual*.