

Harnessing Multiple Inputs and Outputs in the Presence of Limited Buffer Resources

Ali Aqel
11/07/2008

Executive Summary

Many signal processing devices feature a large amount of inputs and outputs but are constrained by a very limited number of buffers for these I/O ports. This application note will show in detail how to make use of multiple input ports and output ports on signal processing devices with limited buffer resources using LabView. The designs in this application note are specific to the Compact Rio 9072 Signal Processing Device. The development described in this note assumes moderate understanding of Electrical and Computer Engineering concepts.

Objective

This application note will explain how to utilize multiple I/O ports on the Compact Rio 9072 in the presence of buffer constraints in LabView.

Introduction

Time synchronized input and output (I/O) is important in any application that involves intensive signal processing. Without such synchronization, digital representations of analog signals will bear no resemblance to their analog counterparts. To achieve time synchronized input and output, buffers are used in order to maintain a constant stream of data from the Analog to Digital (A/D) to the processing unit and vice versa. A common problem with this, however, is the limitations of buffers. Some systems have more inputs and outputs than they have buffers. In the case of the Compact Rio 9072, nine inputs and four outputs are provided with the installation of the NI-9201 and NI-9263 I/O expansion modules while only three First-In-First-Out (FIFO) buffers are provided in hardware. Further observation of the usage of these buffers finds that these buffers are actually underutilized. This underutilization allows for the resources of these FIFOs to be split amongst I/O ports using the LabView Memory block. This application note will describe how to use the LabView memory block to split the resources of FIFOs while maintaining time synchronization.

Hardware

The hardware used in this application note is the Compact Rio 9072 from National Instruments. The 9072 provides three FIFO buffers, eight expansion slots, an onboard microprocessor, and an onboard Xilinx Spartan-3 Field Programmable Gate Array programmable logic device.



Figure 1: Compact Rio 9072

To provide analog inputs and outputs, expansion modules must be used. This functionality is provided by the NI-9201 Analog Input Module and the NI-9263 Analog Output Module.



Figure 2: NI-9201 Analog Input



Figure 3: NI 9263 Analog Output Module

Software

The software used in this application note is LabView 8.6 provided by National Instruments. The software uses a graphical programming language known as G and as such, all development with LabView is done in function blocks.

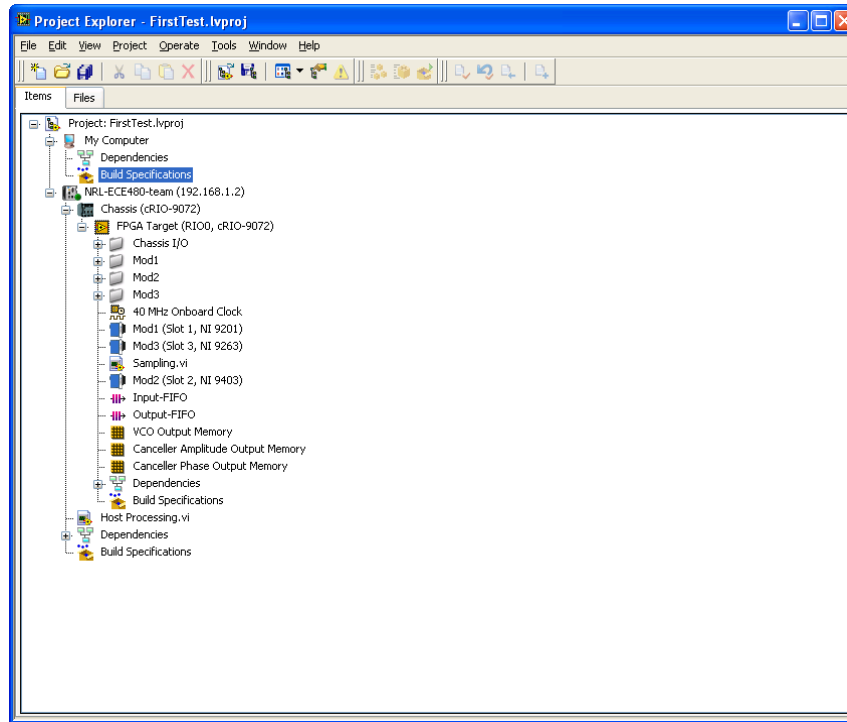


Figure 4: Labview Project Explorer

Implementation

One of the primary goals of this note is to demonstrate how to maintain time synchronization while managing multiple input and output ports with limited buffer resources. In Labview, an object that can be used to achieve time synchronization is the tick counter function block. This block counts to a certain count before returning a done signal. This done signal, however, is useless if the counting is not used to hold execution of a process. To effectively put execution in order (and thus allowing it to be put on hold), a Flat Sequence Structure block is used. The tick counter function block is then placed inside the first frame of the sequence structure as shown in Figure 5. This part of the sequence essentially determines the sampling rate for the entire system.

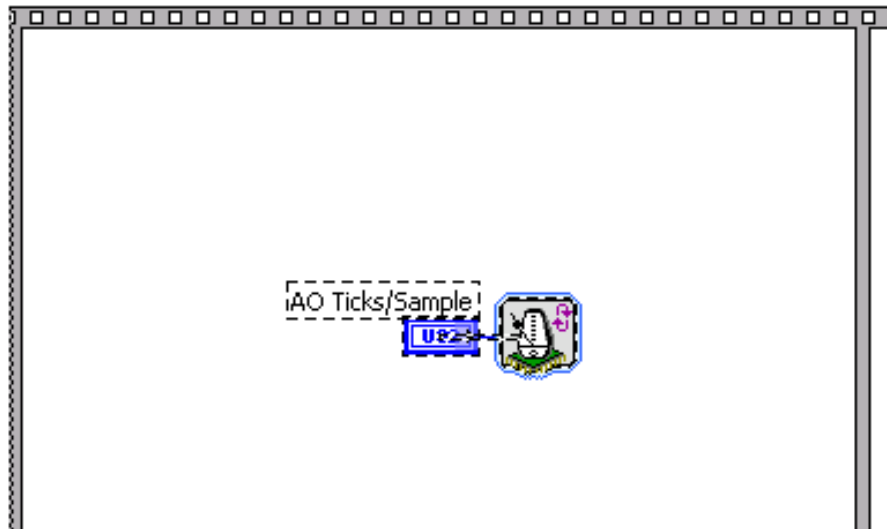


Figure 5: A Tick Counter in Sequence Structure

After execution of the first phase of the sequence structure, the program is now ready to sample. For the sake of simplicity, this note will deal solely with analog output; however, sampling with analog input is done much the same way. In this next phase, the Memory function block is used to transfer data to the function blocks of the respective outputs. In Figure 6, three Memory blocks are used to transfer data to their own respective outputs. This execution step also features logic used to compute whether or not functions found in the next step are to execute. This logic will be explained later.

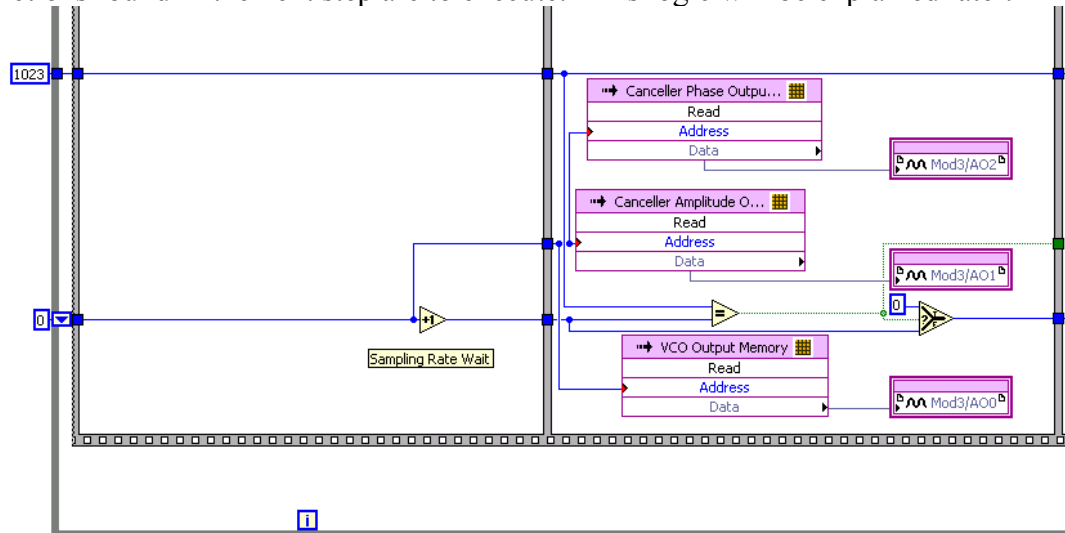


Figure 6: Memory Blocks and Outputs

Figure 6 also shows logic and inputs from the first execution step. The first input, labeled "1023" is used to determine execution of the next step of sequence block not shown above. The bottom input is tied to an increment function block that determines the address of the individual memories from where data is to be pulled. This result is also used to determine the execution of the next step of the sequence structure.

In the third and final step of the sequence structure, a True/False structure is used to determine whether anything is executed. The condition for this structure to execute is for the value of the output of the increment function block to equal 1023. This condition is used to ensure that only 1023 data transfers occur between the memory blocks and the output blocks. Once these 1023 data transfers are completed, an interrupt is fired alerting the software using this sampling system that it is done with this current cycle of samples. It should be noted that 1023 isn't an arbitrary number but a number used for this specific application of a multiple output system. This logic provides for a uniform method of sampling and thus it is vital to maintaining time-synchronization. Figure 7 shows the use of a conditional function block and an interrupt generator to control uniformity of sampling.

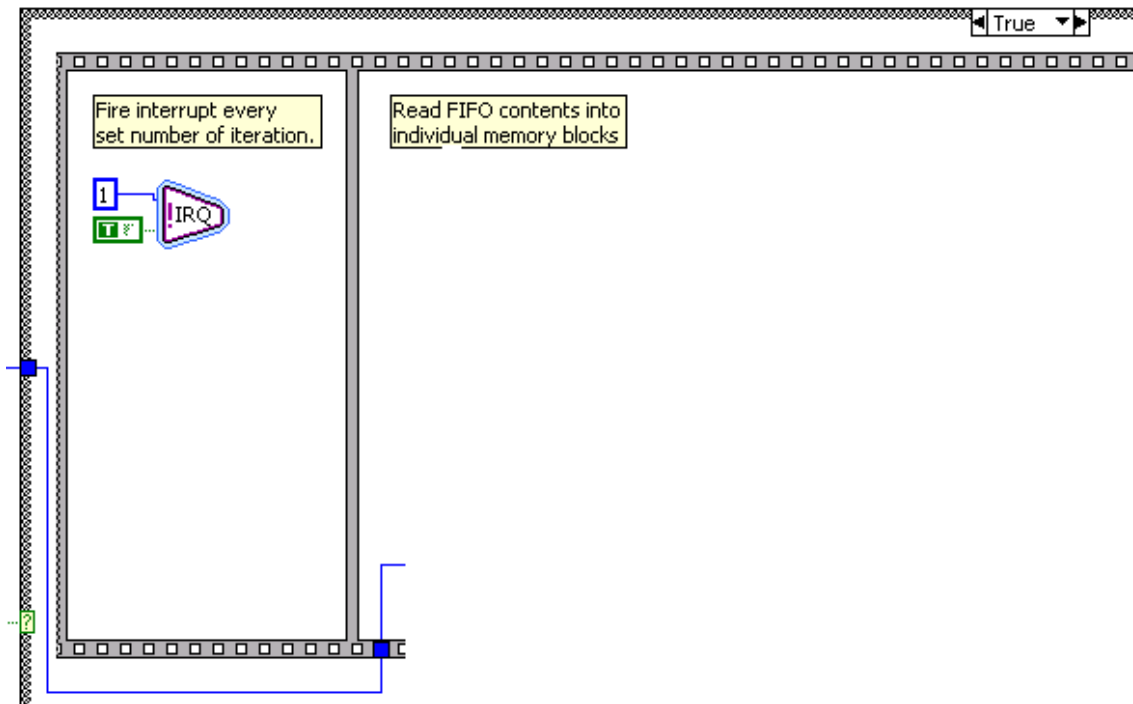


Figure 7: Conditional Block with IRQ Generator

Lastly, the final step to splitting the resources of a single FIFO amongst multiple outputs is to ensure that the memory blocks are filled for the next iteration of sampling. On a Compact RIO based signal processing system, the signal processing software does not see the individual memories or the outputs, but only the FIFO buffer. Thus, when it outputs data, it outputs it directly to a FIFO and the software developed in this note deals with the data found in that FIFO.

One way to split three outputs with their own 1023 samples of data is to put their blocks in separate parts of the FIFO. Thus, the first 1023 samples found on the FIFO go to one output, samples 1024-2046 go to the second output, and samples 2046-3069 go to the third output. In LabView, this can be done with a Case structure and a While loop structure. In Figure 8, the While loop is told to execute 3×1023 times to take care of all samples on the FIFO and the current value of the iteration is passed to the Case Structure.

During the first 1023 samples, the FIFO outputs to the memory of the first output port, as shown in Figure 8.

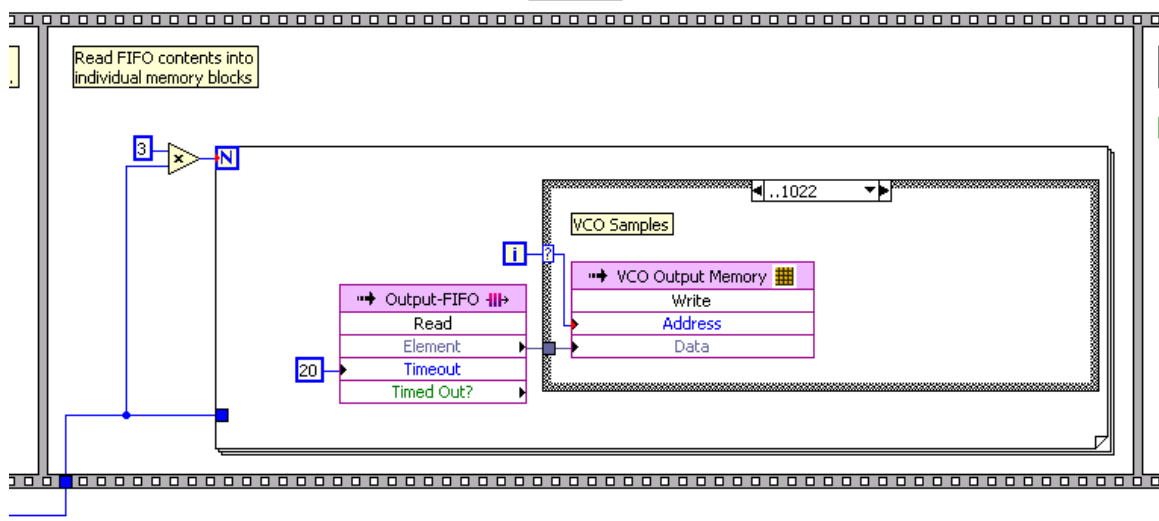


Figure 8: FIFO Outputting to First Memory

After the samples of the first output are transferred, the samples of the second output are then transferred.

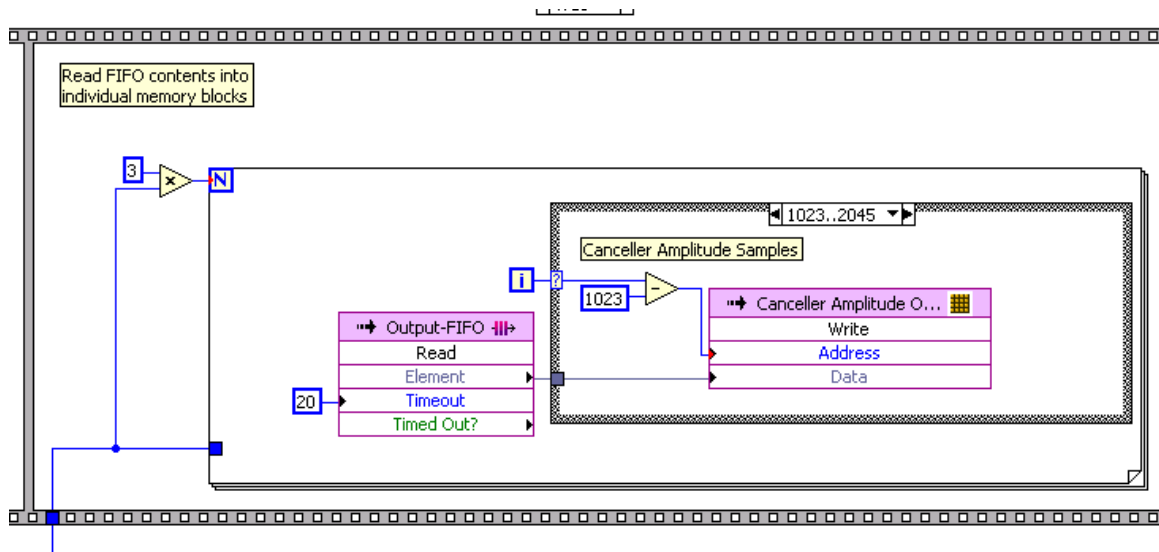


Figure 9: FIFO Outputting to Second Memory

Finally, the samples for the third output and then transferred in the final case.

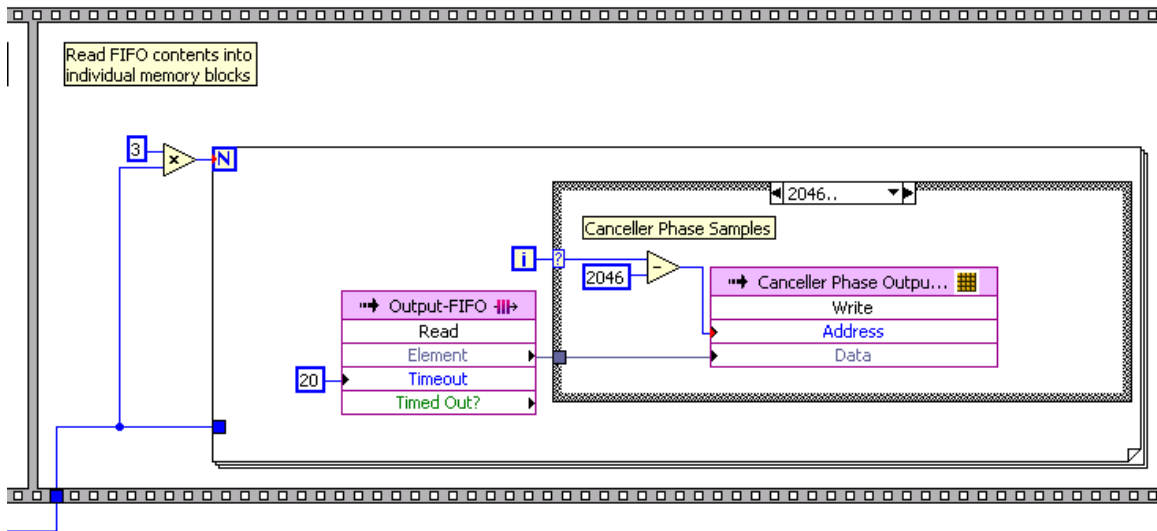


Figure 10: FIFO Outputting to third Memory

At this point, development of software to handle multiple outputs is complete and any LabView signal processing application that calls this sampling program can harness three outputs without worrying about exhausting all three FIFO buffers found on the Compact RIO 9072.

Conclusion and Recommendations

Using the LabView memory blocks and appropriate logic blocks, it is possible to achieve buffer-based time synchronization with multiple input and output ports in the presence of hardware constraints. The code demonstrated results in operation similar to a sampling program exhausting all three FIFOs present on the Compact RIO. One recommendation is to ensure that the program filling the FIFO buffer in a situation like that above understands the separation structure utilized by the sampling program.

References

LabView

<http://www.ni.com/labview/>

Compact RIO 9072 Datasheet

http://www.ni.com/pdf/products/us/cat_crio907x.pdf