SwaMURAy – Swapping Memory Unit for Radio Astronomy

Simon Winberg
Department of Electrical Engineering
University of Cape Town
Cape Town, South Africa
simon.winberg@uct.ac.za

Abstract. This paper concerns design and performance testing of an HDL module called SwaMURAy that is a configurable, high-speed data sequencing and flow control module serving as an intermediary between data acquisition and subsequent processing stages. While a FIFO suffices for many applications, our case needed a more elaborate solution to overcome legacy design limitations. The SwaMURAy is designed around a system where a block of sampled data is acquired at a fast rate and is then distributed among multiple processing paths to achieve a desired overall processing rate. This architecture provides an effective design pattern around which various software defined radio (SDR) and radio astronomy applications can be built. This solution was partly in response to legacy design restrictions of the SDR platform we used, a difficulty likely experienced by many developers whereby new sampling peripherals are inhibited by legacy characteristics of an underlying reconfigurable platform. Our SDR platform had a planned lifetime of at least five years as a complete redesign and re-fabrication would be too costly. While the SwaMURAy overcame some performance problems, other problems arose. This paper overviews the SwaMURAy design, performance improvements achieved in an SDR case study, and discusses remaining limitations and workarounds we expect will achieve further improvements.

1. Introduction
A clear trend in radio processing is the increasing demand for higher sampling speeds, greater resolution, greater bandwidth, and more samples to get more from available parts of the EM spectrum. Going for higher speeds, wider bands, and more feeds means bigger data volumes, in turn needing greater processing power and architectural solutions to support application needs. Combined with this tendency is the increasing popularity of reconfigurable computing (RC) platforms [1] as a means to save costs and development time, among other advantages [2]. A development strain can thus develop between increasing application demands and the design constraints, and aging technologies, of the reconfigurable platform around which an application is built. This paper discusses one such situation and the approach we used to work around limitations of our legacy platform during the development of a Software Defined Radio (SDR) application prototype.

The main reason for our decision to follow a RC platforms approach in our prototyping experiment was the time and cost savings it offered. In particular, a newer and more feature-rich analogue-to-digital convertor (ADC) daughterboard offered potential to extend the application scope [3] and to
explore new innovation potentials by reusing existing RC platform and code developed in a previous project. This led to throughput challenges that the SwaMURAy module was to address.

2. Background and purpose for development of SwaMURAy

Field programmable gate arrays (FPGAs) have become widely used for SDR and radio astronomy applications. A benefit of FPGAs is their potential for providing highly parallel solutions at a lower cost (for small volumes) in comparison to an ASIC approach [4]. In 2009 we embarked on an in-house design for our own open-hardware RC platform to leverage prototyping benefits of this approach. This led to design of the Reconfigurable Hardware Interface for computationN and radiO (RHINO) platform.

Development of RHINO started with a thorough needs analysis followed by an optimized selection of parts and tradeoffs made to minimum costs and maximize compatibility with the better known ROACH platform [5]. RHINO is the legacy platform discussed in this paper.

RHINO was designed around similarity to ROACH version 1 to improve opportunity for code reuse and to facilitate transition from training contexts and small scale and low budget prototyping experiments to larger scale endeavors using ROACH. There are three versions of RHINO, namely RHINO versions 1.01, 1.02 and the Zynq-based RHINO-SDR. The platform related to this paper is RHINO 1.02. An overview of RHINO platform is shown in Figure 1.

Like ROACH 1, the RHINO architectural design is separated into three subsystems: the control processor, the FPGA subsystem, and the monitoring and management subsystem. In the case of RHINO the control processor is an ARM Cortex microcontroller instead of a PowerPC. The RHINO has a Spartan 6 FPGA, the performance and capacity of which is less than the ROACH’s Vertex 5. This Spartan was chosen because, at the time of designing the board in 2010, it was determined to be the lowest cost and best performing option using tools that our design team and our industry partners were familiar with [6].

The root cause for the SwaMURAy occurred during an attempt to prototype an experimental system that would simultaneously run two semi-independent operations on a single RHINO platform. In order to run this application we planned to use of a commercial-off-the-shelf, high-speed sampling board, namely the FMC150 from 4DSP. Previously we had focused on narrower band acquisition and more RF operations happening in hardware, for which sub-10MS/s sampling rates were adequate. The FMC150 can sample two channels, up to 250MS/s. The intended application involved wideband sampling of multiple FM channels (on port A) and sensing solar events on a finely tuned separate channel (port B). Digital down converters (DDCs) were to be used to cut down the data volume and extract individual FM channels sent on to the host PC. The main problem was that the existing processing blocks could not process the data quickly enough; hence a solution for this was needed.
3. Methodology
The major design requirements to be achieved using the SwaMURAy module were: a) sampling at 100MS/s with 14 bit samples, and b) to allow the data to be processed by digital filters achieving at least 20Mb/s post filtering output speeds, i.e. the speed to send partially processed data to the host PC.

The SwaMURAy module was developed by first following a simulation-based approach whereby we developed a simulated ADC interface module which read test signals from a file and mimicked interfacing characteristics of the actual ADC peripheral. Similarly a mimicked Ethernet module was developed to stream processing results to a binary file. Once a working system had been developed and tested in simulation, the process of porting to the target hardware was commenced. This was a lengthy process of revisiting, improving and retesting the simulation and target implementations.

4. Design of SwaMURAy
The SwaMURAy is designed around being a portable gateware module useful as a scheduling and intermediary flow-control subsystem used to split data, sampled at high stream, into two or more separate data blocks or pipelined processing channels. The module has configuration options, set in the code: it can either write to memory, to dual-buffered RAM (DRAM) or to BRAM; or it can send data via a parallel connection to processing modules. The first case is more complex as integration with the external memory banks needs to be done and necessary HDL code needs to be provided for controlling these operations (i.e., using the core generator in the ISE and wiring up the links). The SwaMURAy module is dependent on a ‘gather’ module to which its input lines are wired. The gather module provides an interface wrapper to an ADC driver or another data source that the SwaMURAy will then split into separate output blocks or streams. A block diagram of the SwaMURAy is shown in Figure 2.

The DUGONG module [9], bottom of Figure 2, is an optional HDL state machine to configure SwaMURAy. In this application it can set block size at run time to specify when to swap from sending to one block/stream to another. This setting can alternatively be hard-coded in the SwaMURAy code.

5. Results and conclusions
Initial simulation-based testing was performed using iVerilog, which was limited to functional simulation. A test bench was coded to evaluate SwaMURAy using pre-generated sinusoidal signals and streamed via file to substitute for the ADC input. The SwaMURAy swapped between sending data to two DDC processing modules. The simulation results worked correctly, showing no glitches in the processing caused by potential clocking, synchronization or handshaking mistakes.

Moving to the hardware implementation was more involved. The main challenge was integrating the DRAM control logic. A drawback of the DRAM controller for the RHINO is that it utilizes an intermediary cache, a BRAM buffer, and a state machine controller that accepts a starting address and block transfer size to send/receive data to DRAM. The RHINO has two independently addressable
memory banks for which the read/write transfers needs to be carefully scheduled. A maximum DRAM cache size of 1024 samples (16-bit words for each) was used. Sending data from the RHINO to the PC, and to validate the processing, an HDL Ethernet module was incorporated since the RHINO has no onboard Ethernet peripheral. The OpenCores Ethernet 1Gb/s MAC was utilized to this end.

From the testing, it was found that the desired 20Mbs output via Ethernet from the RHINO to a PC was achieve. However only output from the DDC results of the wideband FM sampled channels were streamed to the PC – the H1 sampled data was not streamed simultaneously but instead had to be multiplexed through the single Ethernet port if the data was required. The FFT for H1 processing was not done in gateware. In performance analysis of this sampling chain of ADC to SwaMURAy to BRAM input of DDC (two DDCs were swapped between) and out to the Ethernet and then to the PC, the maximum sustainable input ADC sampling rate was 61.4MSps, corresponding to a raw data transfer speed of 982Mbps (14 bit sample words padded to 16 bits). Note that this was bypassing the DRAM and having the SwaMURAy write directly to BRAM of the DDC awaiting input, and there was no FFT stage after this on the FPGA; the FFT was done on the host PC. Accordingly, the desired ADC speed of 100MSps with the whole processing chain operating on the FPGA was not fully achieved. Further optimization is needed to get closer to the desired performance level. Furthermore, the SwaMURAy DRAM mode could only operate reliably at slower speeds as data was lost during switchover periods. Nevertheless, simultaneous processing of sampled data ran successfully with SwaMURAy operating as planned for processing the FM channels. In simulation the 100MS/s case with simultaneous H1 processing was successful.

Plans for further work focus at getting the higher sampling rates to work reliably, for which a major problem was synchronizing of data sent from the SwaMURAy to DRAM whereby new data could start overwriting previous memory that had not been fully processed. Any processing module having to re-access data in DRAM was not possible in this design; the FIFO read index could not be reset to allow a second pass through the data; this feature needs to be added. Accordingly, improvement to the SwaMURAy interface is needed to account for these desirable features.

6. Acknowledgement
Thanks to members of the SDRG for input on RHINO, and to the MeerKAT and National Research Fund (NRF) of South Africa for project funding.

References
[1] Todman T J, Constantinides G A, Wilton S J E, Mencer O, Luk W and Cheung P Y K 2005 Reconfigurable computing: architectures and design methods IEE Proceedings-Computers Digit. Tech. 152 193–207
[2] Hauck S and DeHon A 2010 Reconfigurable computing: the theory and practice of FPGA-based computation (Morgan Kaufmann)
[3] Kuo S M, Lee B H and Tian W 2013 Real-Time Digital Signal Processing: Fundamentals, Implementations and Applications (John Wiley & Sons)
[4] Lin M-B 2008 Digital System Designs and Practices: Using Verilog HDL and FPGAs (Wiley Publishing)
[5] Scott S 2011 RHINO: Reconfigurable Hardware Interface for Computation and Radio (University of Cape Town)
[6] Winberg S, Langman A and Scott S 2011 The RHINO platform: charging towards innovation and skills development in software defined radio South African Institute for Computer Scientists and Information Technologists (Cape Town: ACM) pp 334–7
[7] Hamilton B K and So H K H 2010 BORPH: Operating system support on the NetFPGA platform North American NetFPGA Developers Workshop
[8] Hamilton B and Langman A 2011 BORPH port for RHINO
[9] Bridges M 2015 DUGONG (University of Cape Town)