Embedded Linux on FPGA Instruments for Control Interface and Remote Management

Billy Huang[1,2], Graham Naylor[1], Geoffrey Cunningham[1], Olivier Goudard[4], Jordan Harrison[5], Richard Myers[2], Ray Sharples[2], Roddy Vann[3]

[1]EURATOM/UKAEA Fusion Association, Culham Science Centre, Abingdon OX14 3DB, UK
[2]Department of Physics, University of Durham, South Road, Durham DH1 3LE, UK.
[3]Department of Physics, University of York, Heslington, York YO10 5DD, UK.
[4]ESRF, 6 rue Jules Horowitz, Grenoble 38043, France.
[5]Merton College, Oxford, OX1 4JD, UK.

b.k.huang@dur.ac.uk, graham.naylor@ccfe.ac.uk, geoffrey.cunningham@ccfe.ac.uk, goudard@esrf.fr, jordan.harrison@merton.ox.ac.uk, r.m.myers@dur.ac.uk, r.m.sharples@dur.ac.uk, roddy.vann@york.ac.uk.

Funded by EPSRC and EURATOM.

Introduction

- Embedded Linux on an FPGA using the Microblaze soft processor.
- Development tools for embedded linux and firmware development.
- The AXI bus and a fast real-time memory streaming application using embedded linux.
- Remote management using remote firmware updates, network booting and embedded linux controlled watchdog.
- Remote control using ModBus and StarkStream protocols.

Embedded Linux on an FPGA

- Embedded Linux is a powerful tool for system integration. It allows an FPGA systems to more easily be integrated with other control systems whilst reducing the cost of having a more expensive solution with a separate processor system.

- A soft processor is one which is described by a hardware language. This is then “programmed” into the FPGA. This reduces the cost by reducing the number of components on the board.

- A soft processor typically operates at slower frequencies ~50 to 150 MHz. Although you are sacrificing speed you are gaining on system portability and reducing the chances of obsolescence.

- The Microblaze processor is made by Xilinx and Nios by Altera. The are cycle accurate clones of these processor which are open source and run on both manufacturer’s FPGAs – however there are various development issues in using these open source versions.

Embedded Linux Features

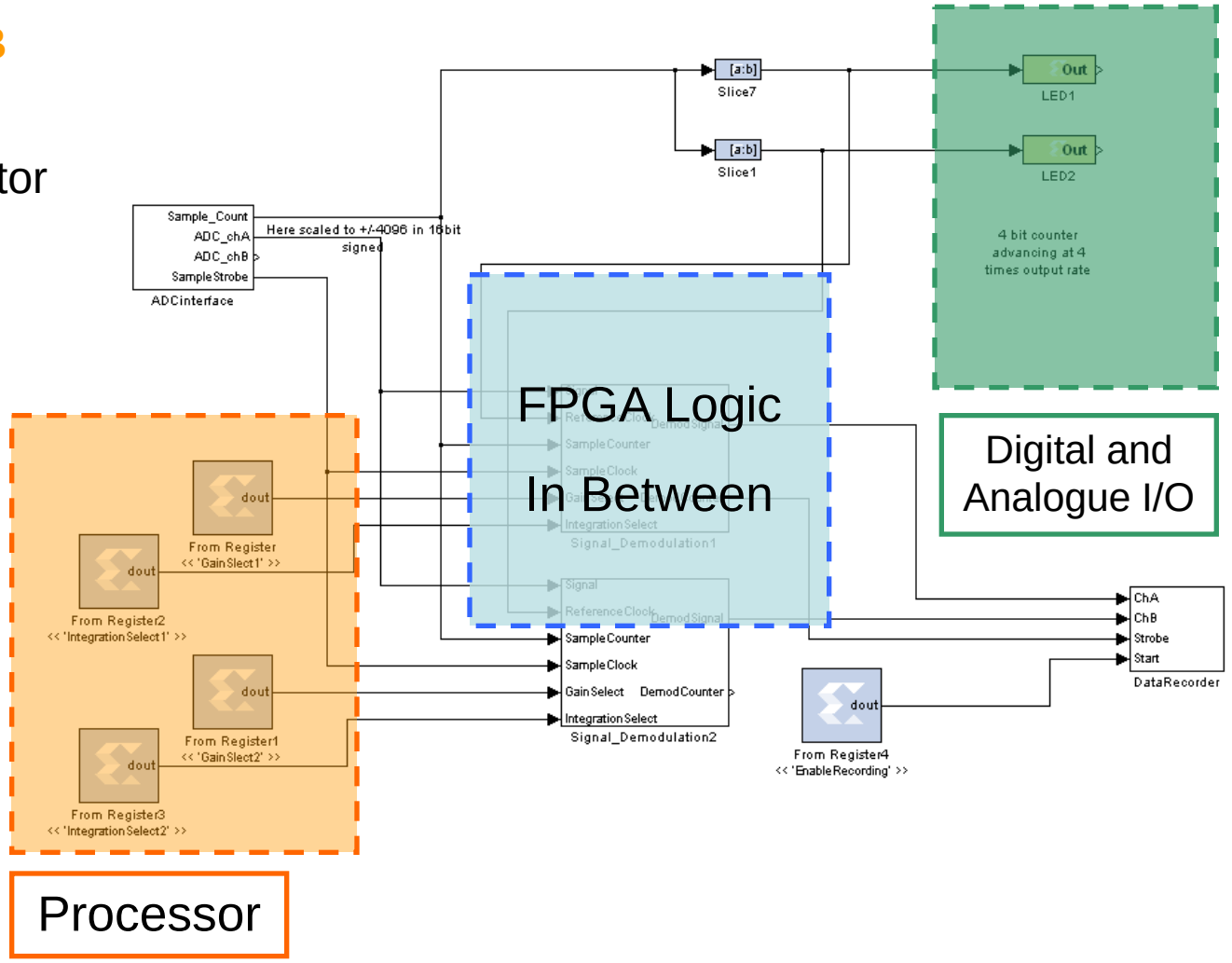
- Network booting using U-Boot allows easier maintenance of an array of systems.
- NFS allows remote software development and testing.
- Mounting of flash drives/chips for firmware updates.
- SSH for manual diagnosis.
- Web interface, ModBus, etc.
- Easy programming for FPGA logic interfacing, through “user space” applications “kernel space”.
- And more...

High level logic programming

Using MATLAB Simulink

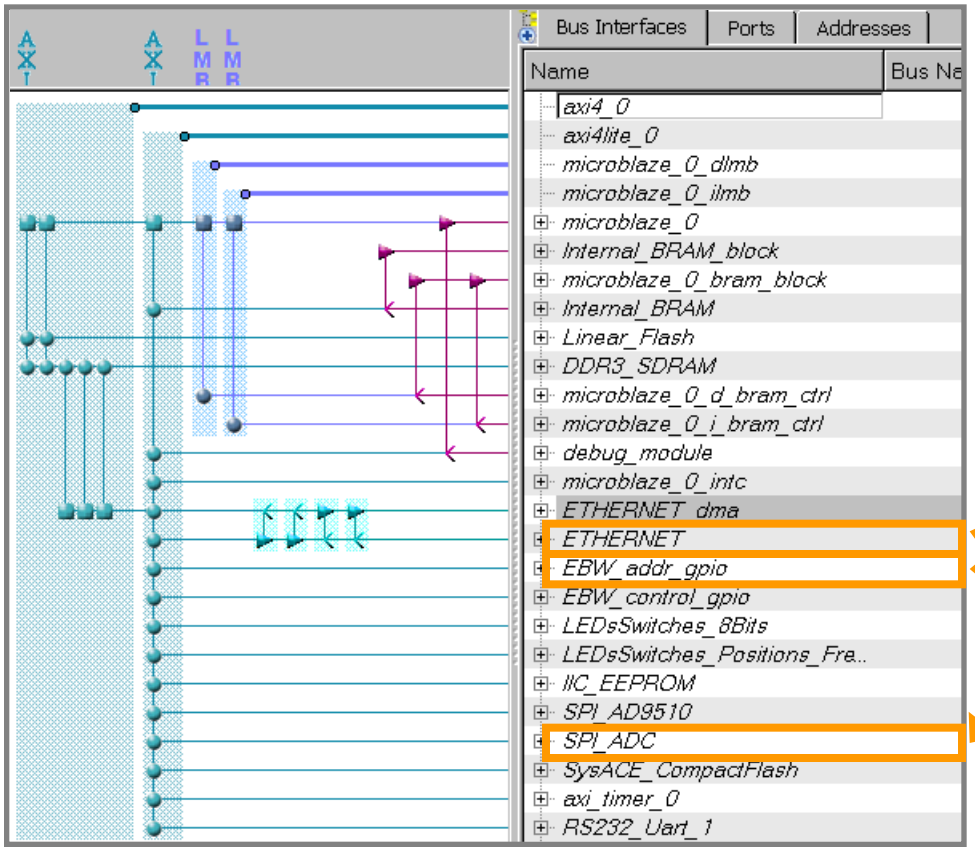
System Generator (Xilinx)

DSP Builder (Altera)



Connecting to the processor

- Linking custom FPGA logic to the processor.



Custom logic created in high level tools

Setting the Registers.

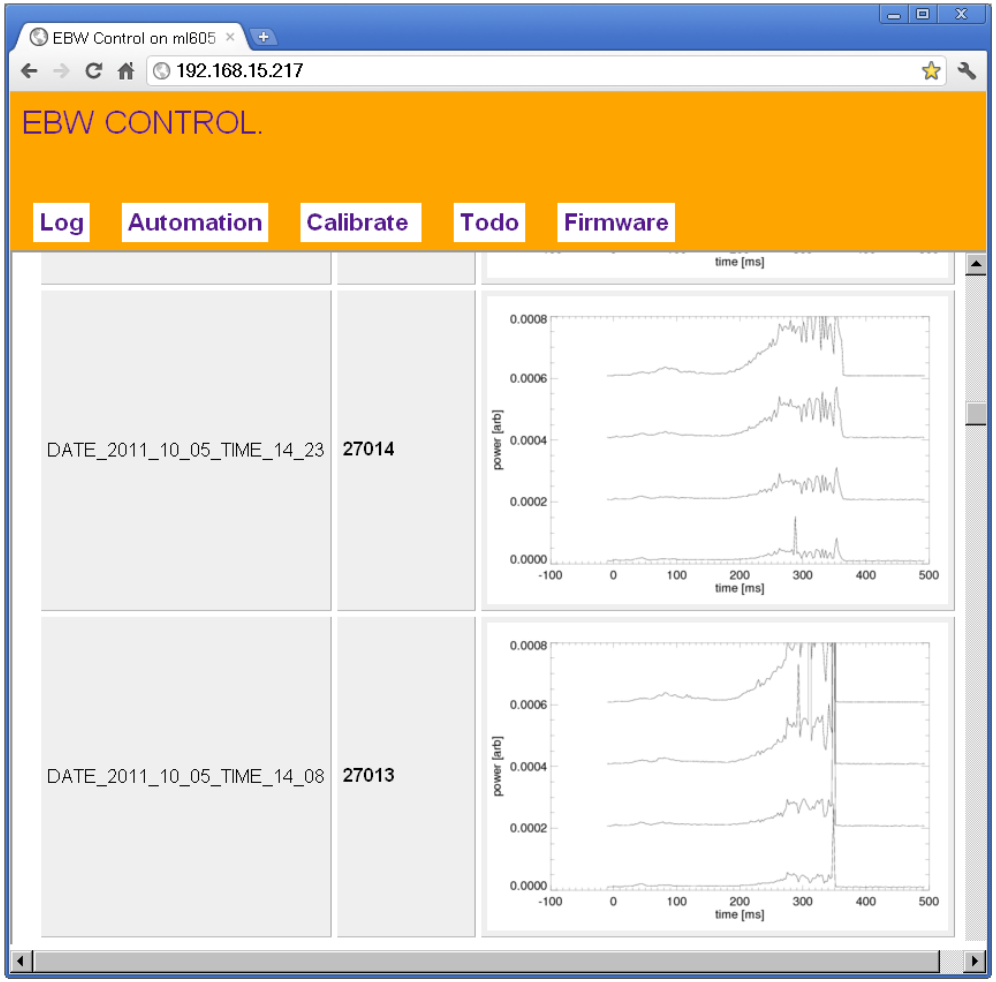
- Setting the shared registers addresses.

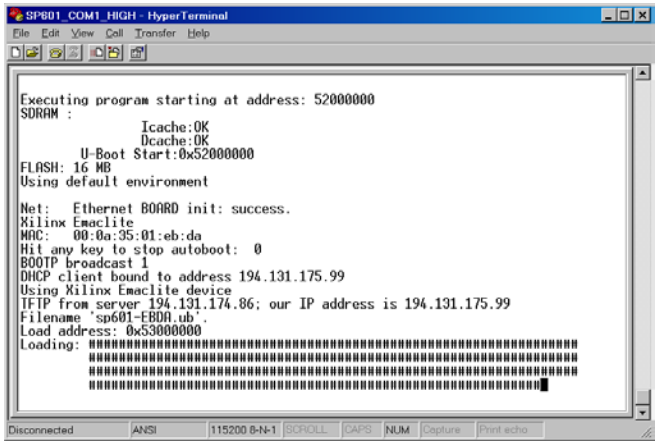
Custom logic with register addresses

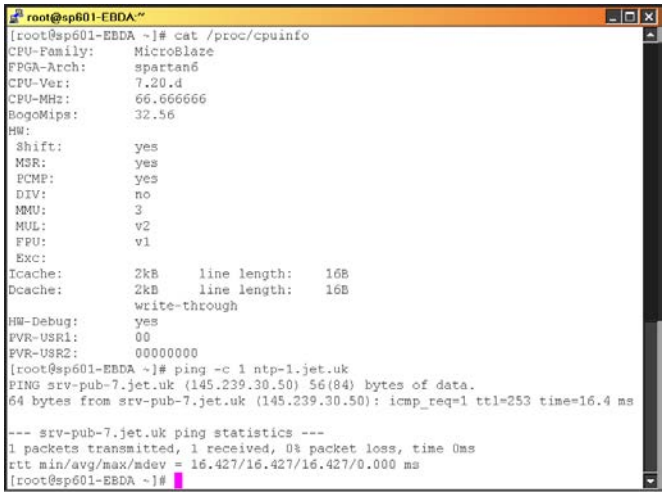
Bus Interfaces | Ports | Addresses

Instance	Base Address	High Address	Size	Bus Interface(s	Bus Name
microblaze_0's Address Map					
microblaze_0_d_bram_ctrl	*0x00000000*	0x0001FFFF	128K	SLMB	microblaze_0_dlmb
microblaze_0_i_bram_ctrl	*0x00000000*	0x0001FFFF	128K	SLMB	microblaze_0_ilmb
LEDsSwitches_Positions_Fre...	*0x40020000*	0x4002FFFF	64K	S_AXI	axi4lite_0
LEDsSwitches_8Bits	*0x40040000*	0x4004FFFF	64K	S_AXI	axi4lite_0
RS232_Uart_1	*0x40400000*	0x4040FFFF	64K	S_AXI	axi4lite_0
IIC_EEPROM	*0x40840000*	0x4084FFFF	64K	S_AXI	axi4lite_0
microblaze_0_intc	*0x41200000*	0x4120FFFF	64K	S_AXI	axi4lite_0
ETHERNET	*0x41240000*	0x4127FFFF	256K	S_AXI	axi4lite_0
SysACE_CompactFlash	*0x41800000*	0x4180FFFF	64K	S_AXI	axi4lite_0
axi_timer_0	*0x41C00000*	0x41C0FFFF	64K	S_AXI	axi4lite_0
ETHERNET_dma	*0x41E00000*	0x41E0FFFF	64K	S_AXI_LITE	axi4lite_0
SPI_AD9510	*0x42000000*	0x4200FFFF	64K	S_AXI	axi4lite_0
SPI_ADC	*0x42020000*	0x4202FFFF	64K	S_AXI	axi4lite_0
EBW_control_gpio	*0x42040000*	0x4204FFFF	64K	S_AXI	axi4lite_0
EBW_addr_gpio	*0x42060000*	0x4206FFFF	64K	S_AXI	axi4lite_0
debug_module	*0x44800000*	0x4480FFFF	64K	S_AXI	
Linear_Flash	*0x70000000*	0x71FFFFFF	32M	S_AXI_MEM	axi4_0
Internal_BRAM	*0x78000000*	0x7803FFFF	256K	S_AXI	axi4lite_0
DDR3_SDRAM	*0x80000000*	0xFFFFFFFF	2G	S_AXI	axi4_0

Embedded Linux User Interfaces







AXI Bus for real-time streaming

- The AXI Bus is an on-chip bus with very fast with 51.2+ Gbps pathways (256 bits wide at 200MHz). These can be used to stream data into a memory DIMM, to transceivers for external communication or other hardware..

- AXI is well developed and could be integrated in future with the ARM processor, such as anticipated for the ZYNQ FPGA using a dual core 1GHz ARM processor.

High speed real-time memory streaming

- We have developed a 64 Gbits/sec system in a ~5U rack system costing ~$10,000 in hardware. We stream 16 channels of 14-bits at 250 Msps into 4GB of memory in 0.5 seconds.

StarkStream: A UDP streaming protocol

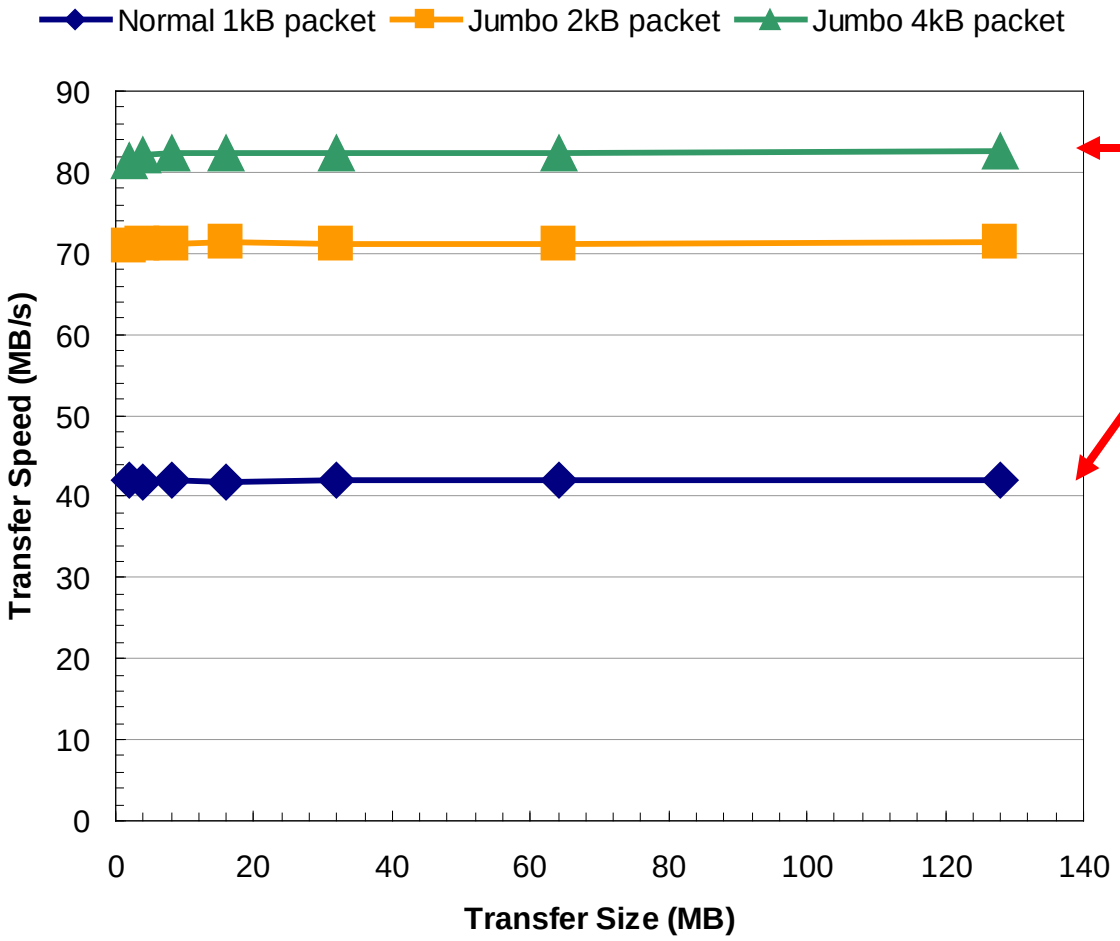
- StarkStream is a low latency UDP protocol designed for streaming data and register control. It operates inside a DMA enabled ethernet driver allowed an embedded processor to achieve very high data rates.

- StarkStream uses jumbo packets which are larger than standard (1500 byte) packet up to ~9000 bytes. Having bigger packets increases throughput but all hardware supports these packets.

- We conduct a UDP streaming test from an FPGA running embedded Linux, which has DMA capable network driver.

- Raw streaming speed of Jumbo packets allow speeds up to 80+ MB/s with packets of size 4096bytes. So by using jumbo packets we have tested a doubling in throughput speed.

UDP Streaming using Jumbo Packets



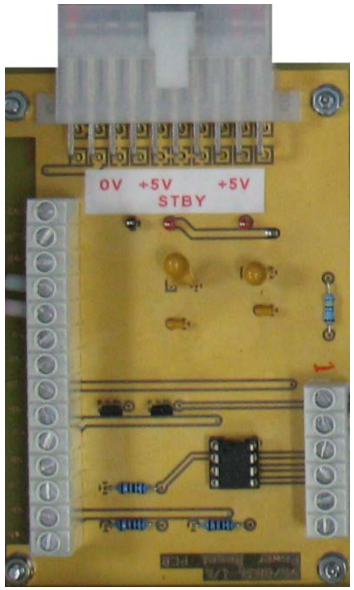
83 MB/s raw throughput

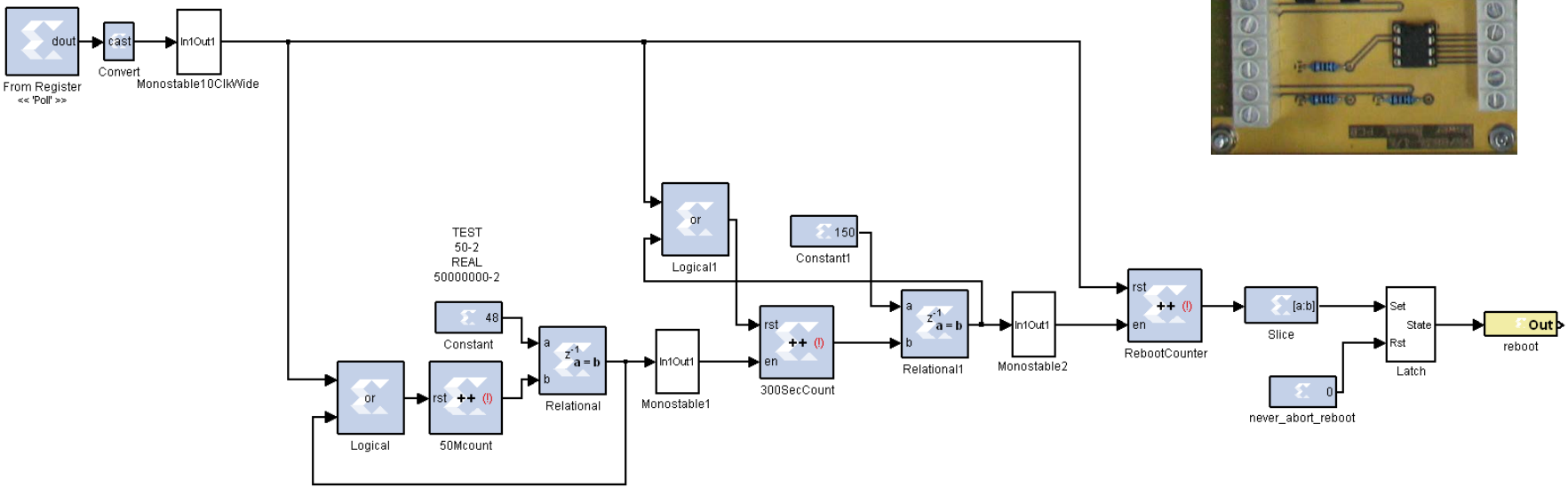
42 MB/s data transfer

- Using embedded Linux and UDP streaming: 129us +/- 13us (or ~10% latency jitter) PC to FPGA.
- FPGA to FPGA latency should be significantly lower but to be tested.

Full remote reboot control

- Optical Hard Reboot
- FPGA Digital I/O Reboot
- Watchdog Reboot

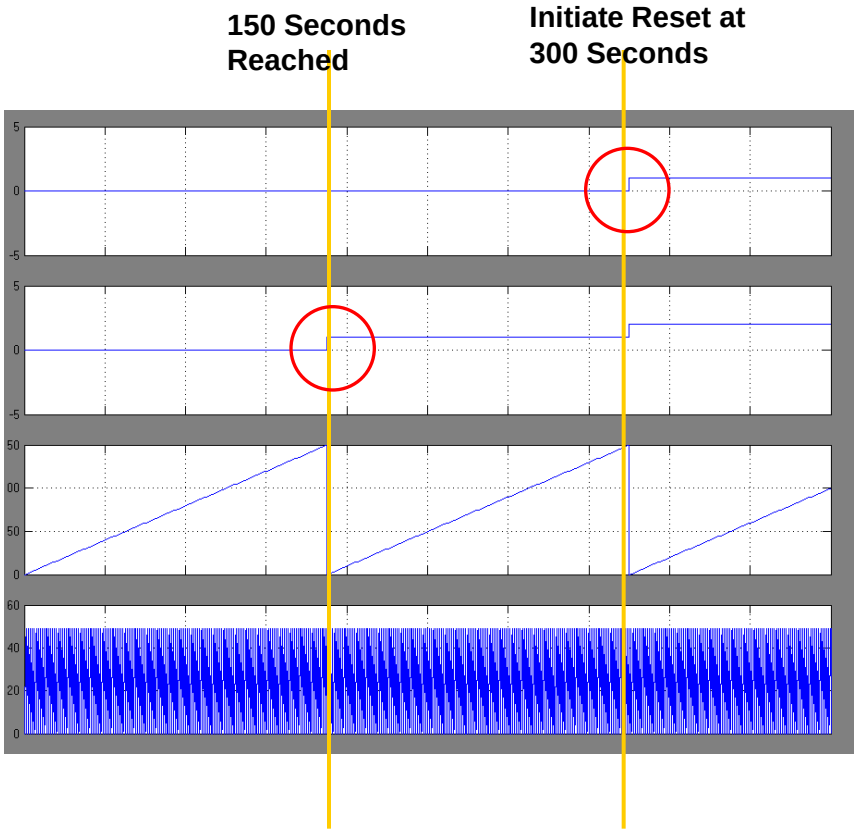




Watchdog

arch/microblaze/kernel/heartbeat.c

```
...

#define POLL_BASE_ADDRESS (0x42080800)

static unsigned int base_addr;

void heartbeat(void)
{
        // Create a pulse to reset the watchdog.
        out_be32(base_addr, 0);
        out_be32(base_addr, 1);
        out_be32(base_addr, 0);
}

void setup_heartbeat(void)
{
        base_addr = POLL_BASE_ADDRESS;
        base_addr = (unsigned long)
ioremap(base_addr, PAGE_SIZE);
        printk(KERN_NOTICE "Watchdog at 0x%x\n",
base_addr);

        // Set gpio output true
        out_be32(base_addr + 4, 0);
}
```



An embedded Linux driven heartbeat poll resets the counter thus preventing a logic pin going high – causing a reboot.

Remote Firmware Updates

- Remote firmware updates allow an array of FPGA systems to be remotely managed and upgraded.

- We accomplish updates by calling a web page on the FPGA board. This runs a script which downloads the firmware from the server and performs md5sum verification with fallback capabilities.

- Problems can occur if writing the firmware and a power failure occurs (or other) which causes a firmware write interruption.

ModBus

- ModBus is a PLC protocol that can be used for FPGA control, but has certain limitations on latency and data throughput.

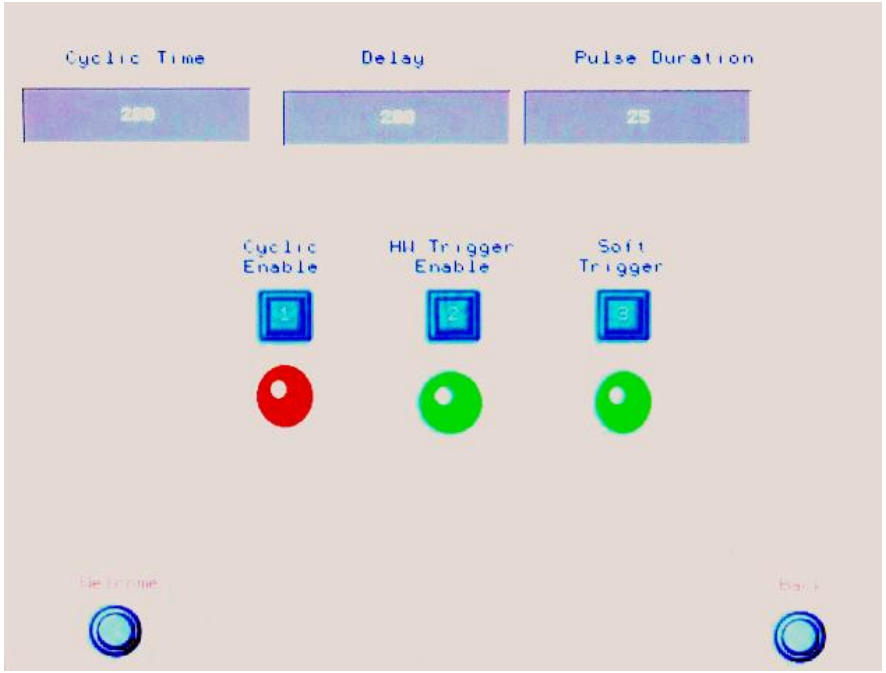
- A modbus server has been created running on Embedded Linux on the Microblaze processor running on a Spartan 6 FPGA.

- The FPGA hardware device is controlled by modbus communication from a PLC master.

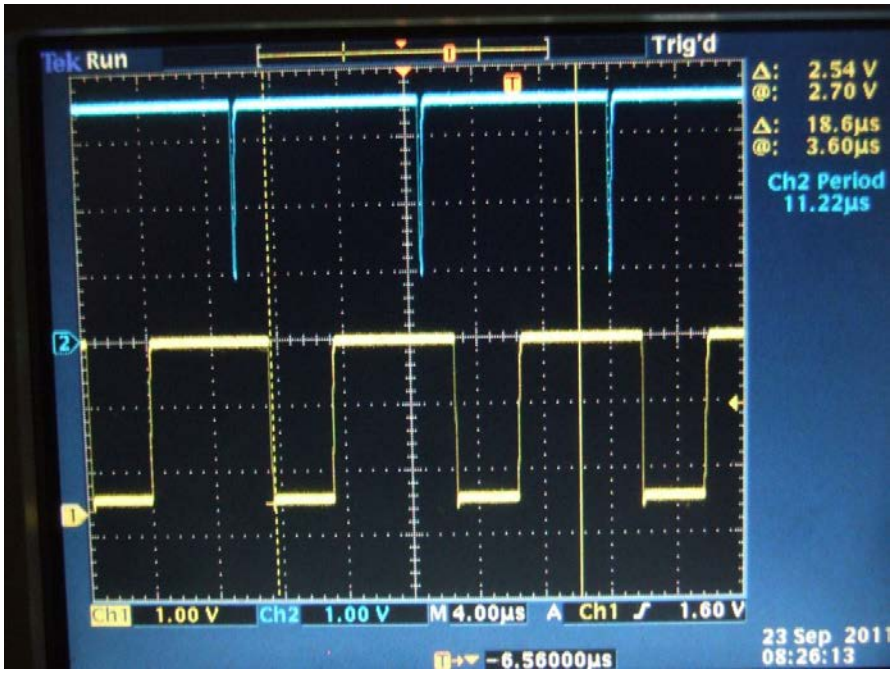
PLC ModBus Control

- PLC controlling the FPGA Modbus system – configuring triggers, pulse delay and duration.

PLC Master

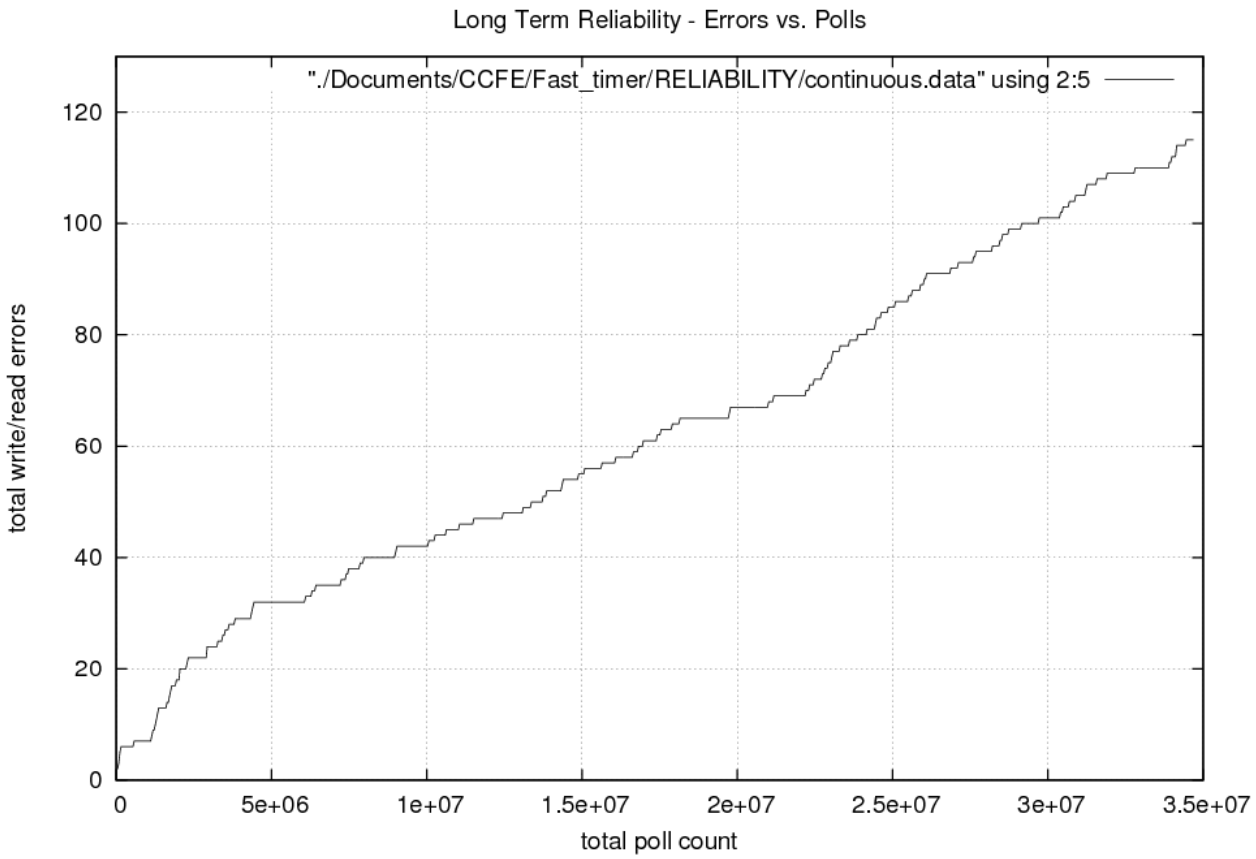


FPGA Modbus Slave System



Modbus Long term errors

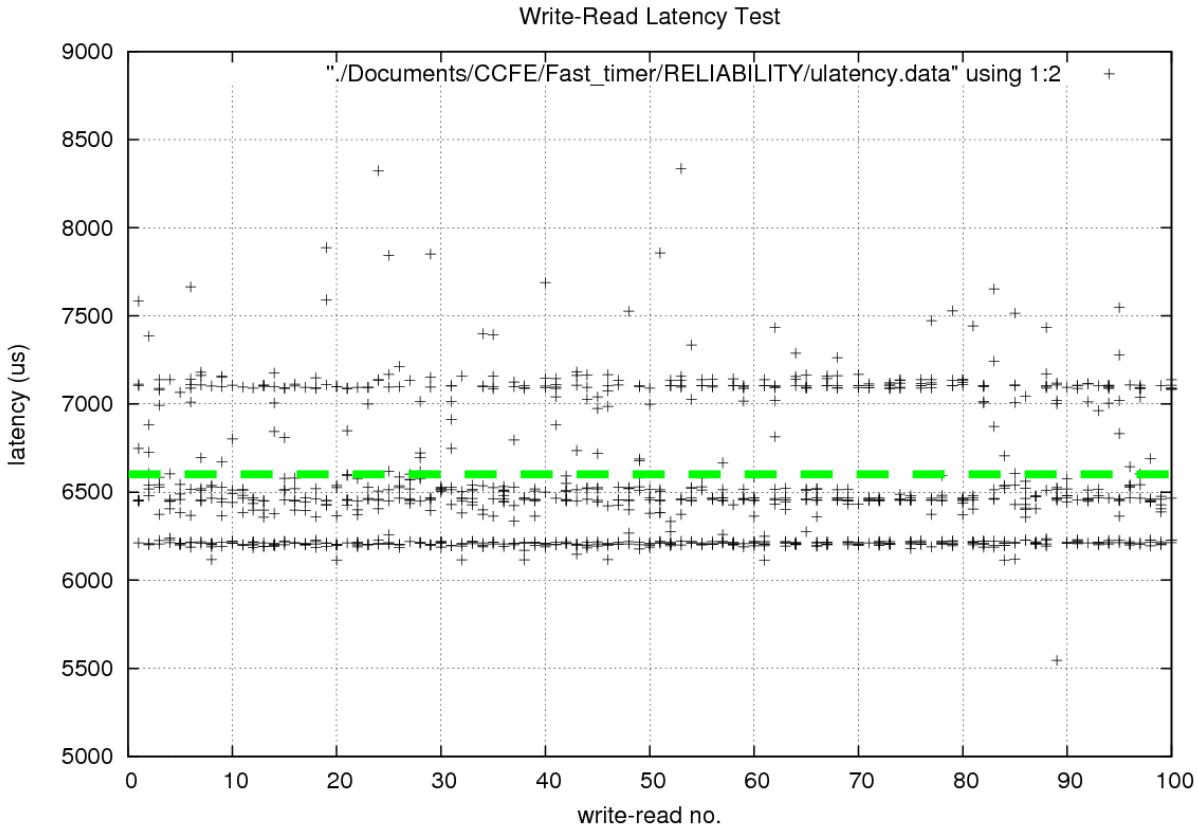
Error ratio ~ 3.4×10^{-6} errors per write/read. An error is not a system failure – Modbus will automatically perform the write or read again.



Modbus Latency

Without outliers (995 samples):
Mean: 6.6 ms
Deviation: 0.38ms

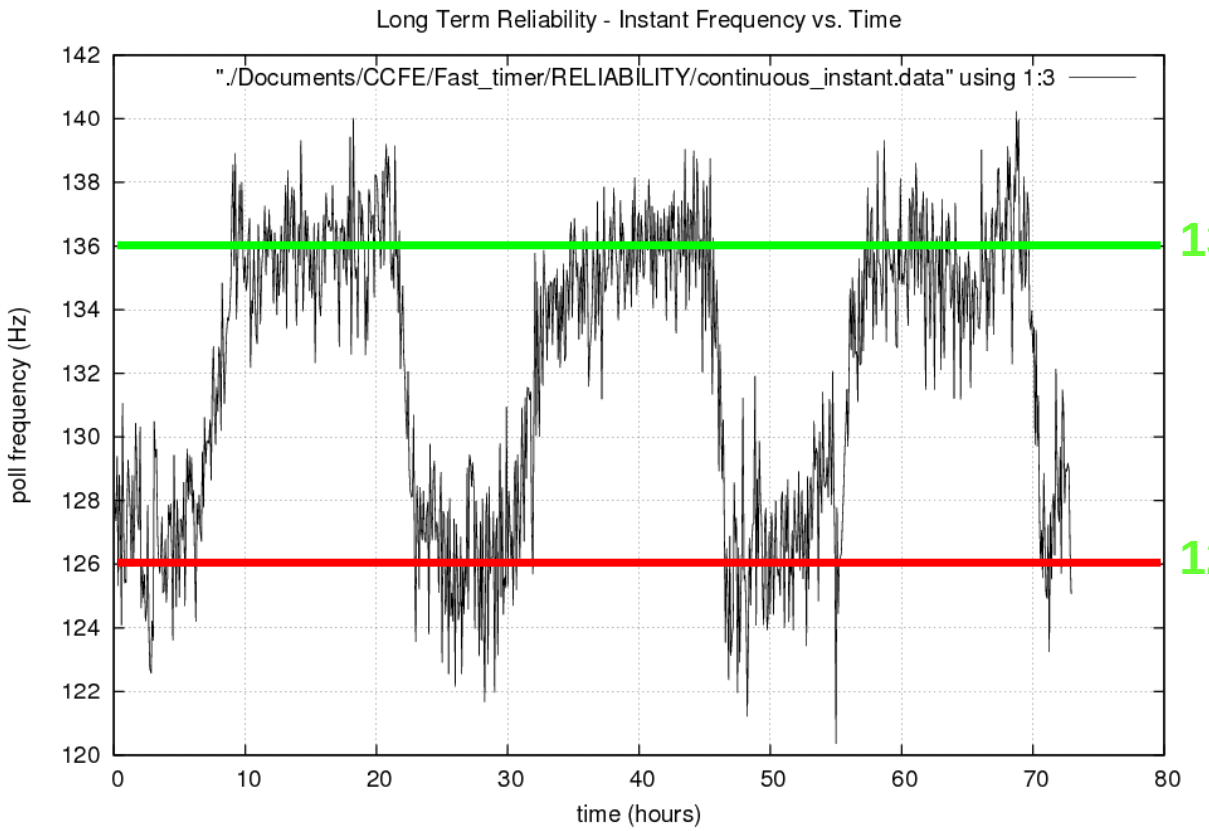
With outliers (5 out of 1000 samples):
Mean: 7.6ms
Deviation: 14.3ms



Mean – 6.6ms

Modbus reliability test over 3 days

Long term polling stability between 126Hz (8.0ms) and 136 Hz (7.4ms)



136 Hz (Night)

126 Hz (Day)

TANGO Control System

- Since the FPGA is running embedded Linux it is potentially possible to run TANGO directly on the FPGA. The advantage of this is that an intermediary software system between the FPGA system and the controller is not needed.
- Due to the slow speed of the soft processor (200MHz) TANGO may not be fast enough to run directly on the FPGA. Currently it crashes at certain stages do to unknown reasons.
- Significant development effort went into modifying the Linux kernel to create a native gdb debugger to resolve these issues but up until now TANGO is not yet running directly on the FPGA.

TANGO/EPICS Control Strategy

- For slow system updates and register control we propose using Modbus enabled Tango and Epics.

- StarkStream for memory streaming, reflective memory and low latency (<100 us) register control. We plan to develop the StarkStream protocol to run under Epics and Tango.

Development Strategy

- Use the appropriate synthesis tools to fix portions of the FPGA firmware, include a standard GPIO interface for most register control on digital I/O.

- Develop a tool to easily drag in the processor and Linux system to simply the development process.

Summary

- Used embedded Linux as a powerful tool to create tightly integrated software/hardware systems.
- Remote management is aided by using embedded Linux controlled watchdogs, remote reboot board, network booting and development.
- Remote control interface using ModBus at sub 6.6ms cycle times and streaming at 40+ MB/s using StarkStream (~129us latency) on a 83 MHz Microblaze.
- Efforts to reduce system development time and improve integration such as testing StarkStream and Modbus under TANGO and EPICS.

FPGA Workshop 2012

- Building on last year, we will be running another hands-on FPGA Workshop at CCFE, Oxfordshire, UK in early February 2012. It will include embedded Linux and FPGA development. Please email billy.huang@ccfe.ac.uk if you are interested in joining.

Thanks

- Thanks to all the authors and to you for listening!

- More info at:
- http://www.ccfe.ac.uk/fpga