Introducing Die Datenkrake: Programmable Logic for Hardware Security Analysis

Dmitry Nedospasov
FG SecT, TU Berlin
dmitry@sec.t-labs.tu-berlin.de

Thorsten Schröder
modzero AG
ths@modzero.ch

Abstract

This work presents Die Datenkrake, an open source hardware USB peripheral for hardware analysis. Die Datenkrake is comprised of an ARM microcontroller and a Field Programmable Logic Array. The design of Die Datenkrake overcomes many limitations that are common to widely used embedded hardware analysis tools. The programmable logic makes it possible to add additional functionality to the ARM MCU such as additional I/O interfaces, support for proprietary protocols and real-time signal processing in hardware. This work also presents several example applications that can greatly benefit from utilizing such a platform versus standard tools.

1 Introduction

In the world of software security, many tools for binary analysis are readily available. Many popular binary analysis tools, such as IDA Pro [4], provide extensible plug-in architectures. Any functionality missing in the core application itself can be implemented as a plug-in. The extensibility of these tools has made them particularly popular. Moreover, entire ecosystems of plug-ins and extensions already exist. Plug-ins can make it possible to seamlessly export data or interface to other programs.

In embedded security the picture is very different. Interfacing with the target device can be particularly tricky. Embedded hardware analysis may require the simultaneous measurement of several signals. For simple applications, standard measurement equipment may suffice. In fact, modern measurement equipment such as digital storage oscilloscopes and logic analyzers provide protocol decoders for many standard embedded protocols. Segmented memory ensures that only traces of relevant data are captured by the hardware. The problem arises when analyzing proprietary protocols that are not included in the device's set of standard protocols.

Several open source projects are built around programmable microcontrollers and serial interfaces. Such platforms commonly include a serial bootloader to facilitate firmware upgrades via the USB to serial interface without the need for an external programmer. One of the most popular projects is the Arduino platform. The Arduino is especially popular because of the availability of expansion boards known as "shields" for the characteristic Arduino footprint. Shields are available that supplement common interfaces and functionality, including radio interfaces, such as Bluetooth and WiFi. The programmable pins of the Arduino make it possible to implement low-speed serial protocols via bit-banging. It is possible to perform common embedded hardware security analysis tasks, such as interfacing to or eavesdropping on embedded memories and interfaces. One of the most notable examples of such an Arduino-based project is JTAGEnum [6], a powerful Arduino sketch for enumerating and detecting JTAG interfaces.

One of the simplest and most versatile tools for embedded analysis is the Dangerous Prototypes Bus Pirate [11]. Unlike the Arduino, the Bus Pirate is commonly available with a simple set of test leads to connect to headers on embedded devices. The connector on the Bus Pirate itself is a standard shrouded 10-pin header that makes it trivial to make custom cables to interface to other devices. The code running on the Bus Pirate hardware provides support for various common embedded buses and interfaces. Incoming data from external interfaces is processed and decoded by the microcontroller and the data is sent via the serial interface to the PC. Such a configuration makes it possible to implement a user-facing Command Line Interface (CLI) to switch between different modes of operation.

However, both the Bus Pirate and Arduino lack the real-time performance necessary for interfacing to high-speed memory interfaces. In his seminal work, "Hacking the XBOX" [5], Andrew "Bunnie" Huang details the process of interfacing to the Hypertransport bus on the PCB.

To capture the high-speed signals on the device, a buffering circuit was used to convert the differential signaling to single ended I/O. Subsequently signals were sampled and buffered in a FIFO implemented on an Field Programmable Gate Array (FPGA). The setup utilizes the advanced I/O configurations that are available on modern FPGAs to demultiplex and capture dataflows that exceed the speed of communication link to the PC.

Though low-cost FPGA and CPLD development boards have become readily available in recent years, they remain fairly unattractive for embedded analysis. Most development boards are not tailored for the implementation of custom interfaces for embedded analysis as development boards commonly supply only high-density I/O expansion connectors. Procurement of such connectors can be very difficult and they require adapter PCBs to access individual pins. Additionally, development boards generally provide only a single programmable device. Using solely an FPGA for such applications makes it difficult to efficiently load and change parameters of registers implemented on the FPGA. By instead integrating a microcontroller alongside the FPGA, the microcontroller can be interfaced directly to the FPGA's bus and used for the configuration internal registers of the FPGA.

High-end test and measurement equipment generally runs full-blown operating systems and high-performance processors. Standardized interfaces, such as VISA [8], allow users to access capture data directly so that protocol analysis can ostensibly be performed on a PC after capturing the data. However, very few manufacturers of test and measurement equipment provide an extensible interface to the actual measurement hardware. Without direct hardware access, real-time analysis becomes particularly difficult to implement efficiently or at all. One interesting solution is the National Instruments FlexRIO [7]. The FlexRIO is an expansion board for the National Instruments PXI backplane system that adds programmable logic for implementing custom I/O interfaces. The system implements the necessary interfaces for communication with the PC. Hence, users must only supply the programmable logic to implementation for the custom I/O that interfaces. Characteristic to National Instruments the system requires the hardware to be programmed in National Instrument's LabView programming environment. This makes this solution particularly unattractive as it adds significant licensing costs for users outside of universities.

In this work we introduce Die Datenkrake[1] (DDK for short), a low-cost extensible hardware analysis platform. We highlight several example applications where the advent of programmable logic makes analysis significantly easier. The DDK utilizes an ARM Cortex-M3 microcontroller (MCU) and an FPGA to provide maximum flexibility for embedded applications. The ARM MCU is connected to a PC via a USB-to-serial interface and is also connected to the FPGA via a parallel bus and multiple serial interfaces. The FPGA provides programmable logic and an interface to eight channels of General Purpose I/O (GPIO). Each GPIO channel consists of six individually configurable bidirectional pins. Internally, the DDK uses a common open-source bus, making it particularly easy to integrate existing logic implementations. Additional I/O protocols can be implemented for the GPIO channels as well as additional logic for real-time data analysis. The DDK is open-source hardware – the hardware design files, as well as the HDL and C code are completely open-source and freely available on the project website [3].

[1]Datenkrake - literally "Data Octopus", is German slang for organizations with questionable privacy practices that hoard and resell user data. The platform consists of eight user-configurable channels for interfacing to external data sources, hence the name.

Organization

The rest of this paper is structured as follows. Section 2 illustrates several hardware security applications that can strongly benefit from programmable logic. Section 3 covers the design of Die Datenkrake and describes essential features missing in other platforms. Further refinements for future generations of the DDK are presented in Section 4. Finally, we conclude with a summary of what is necessary for embedded hardware analysis in Section 5.

2 Example Applications

Programmable logic offers many benefits over general purpose programmable microcontrollers. By utilizing these advantages, targets can be analyzed more efficiently. This section covers three embedded security examples that highlight the advantages of programmable logic. Specifically, analysis of multiple target devices in parallel, applications with advanced timing constraints and timing critical real-time signal processing scenarios are presented. The DDK provides eight identical, but individually configurable channels each with six bidirectional GPIO pins. For simplicity, only a single channel, `chX`, is illustrated in each of the following examples, see Figures 1, 2 and 5.

2.1 Hardware Interfacing

One of the greatest challenges in embedded device analysis is interfacing to the devices themselves. Embedded devices generally provide only limited low-speed serial communication interfaces. However the effective

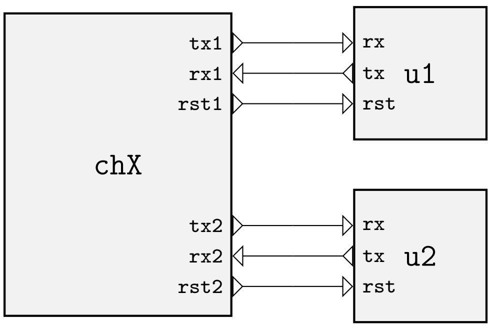

Figure 1: Odroid channel module

throughput to such devices can be increased by interfacing to multiple device instances simultaneously. One basic application is brute-forcing firmware passwords for service or administrative accounts. Successful logins can be automatically identified automatically based on the delay and serial output. By brute-forcing multiple devices in parallel the penalty for failed login attempts can also be reduced.

A more advanced setup may instead fuzz software components by providing test vectors to the Device Under Test (DUT) via the serial interface. Frameworks, such as Peach and Radamsa, already exist for test case generation [10, 12]. However, fuzzing software components and operating systems on embedded devices can lead to crashes that render the DUT entirely or partially unresponsive. It is essential to check whether the DUT is performing erratically during the fuzzing process. When an irregular operating state is detected, the DUT can be reset and brought back to a nominal operating state. Though modern embedded devices may include other interfaces, it is common for such devices to include serial interfaces for debugging purposes. Kernel error messages, for example, are often solely transmitted over the device's serial interface.

As a Proof-of-Concept (PoC), we implemented a simple channel module for the DDK that monitored serial communication. Each channel interfaced 3 signals per device, a bidirectional UART (`tx` and `rx`) and a reset line (`rst`), see Figure 1. Because only three of the six signals provided by the channel are used, two device instances (`u1` and `u2`) can be monitored simultaneously. For our PoC, the DDK was configured to monitor an Odroid-U2[2] embedded single board computer for system crashes. In this configuration, the DDK sent carriage return characters to the target device. Hence, the absence of a carriage return coming back from the target within a timeout window indicated a system crash. An automatic system reset is triggered by the channel module by cutting the supply voltage of the target device via a pass transistor. This implementation was completely realized in the programmable logic of the FPGA without utilizing the ARM microcontroller.

By also utilizing the ARM microcontroller, a more advanced implementation would automate the process even more. The bootloaders of many embedded devices facilitate device upgrades over the serial interface. Hence a new kernel or ramdisk can be passed to the device at boot time over UART. On the DDK, the programmable logic of the FPGA makes it trivial to implement signal multiplexers. In a purely combinational logic implementation, the `tx` pin of the LPC can be multiplexed to the target devices with the parallel `data` bus providing the necessary select signals. The reset lines of the target devices can then be multiplexed in a similar fashion. However, such a setup would not allow simultaneous communication with multiple devices.

Simultaneous logging and monitoring of multiple targets can be realized by utilizing the ARM microcontroller in conjunction with a sequential logic implementation of the FPGA. Error messages decoded by the individual channel modules in the FPGA can be buffered in FIFO memory ensuring that no error events are missed by the system. Upon detecting a system crash, the DDK's LPC ARM microcontroller is alerted to the event over the interrupt line. The LPC reads registers on the FPGA to determine which of the target devices has crashed. A UART multiplexer implemented on the FPGA is configured by the LPC to pass data directly to the target device. The LPC is then configured to pass data from the PC directly to the FPGA. An arbitrary sequence of escape characters can be used to reset the communications once the transfer is completed. Note, in such an implementation parallel channels would continue to be monitored even when a device upgrade is in progress.

2.2 Hardware Glitcher

Secure systems are susceptible to several classes of transient faults that can be introduced during runtime [1]. Transient faults are non-invasive and temporary in nature, but require a high degree of timing precision to execute successfully [2, 15]. Faults mainly induce rise- and hold-time violations within the circuit as well as adversly affecting the threshold voltages within memories. All of these factors can lead to incorrect values being loaded into registers. Transient faults must be executed within a single clock cycle of execution of the target device. Hence, precise timing on the order of fractions of a clock cycle is needed.

Two common forms of transient faults that can be introduced into almost any system are voltage supply and clock glitching. The goal of a voltage glitch is to ad-

[2]`http://www.hardkernel.com/renewal_2011/products/prdt_info.php?g_code=G135341370451`

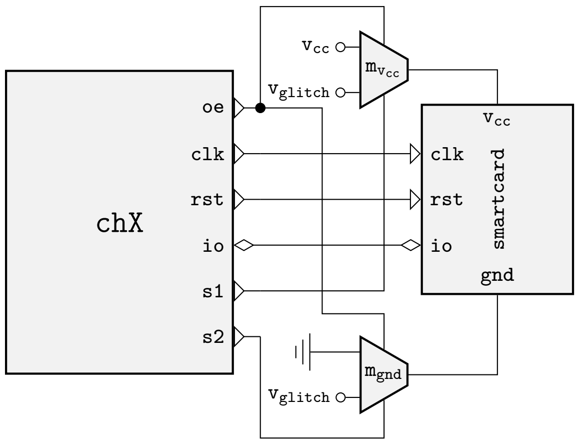

Figure 2: Hardware Glitcher

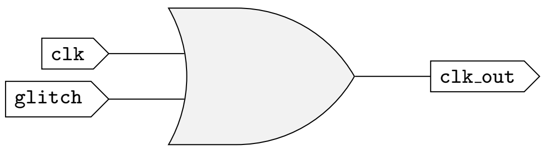

Figure 3: Example OR-gated output clock

versely affect the operating conditions of an IC. In this case the result of a combinational circuit arrives after the sampling occurs, resulting in incorrect intermediate results. Alternatively, a high-voltages read from a devices Non-Volatile-Memory may be corrupted. In clock glitching, the clock period is altered during a given clock cycle. Here the combinational circuit is sampled too early before the correct signal arrives, also resulting in incorrect intermediate results.

FPGA's offer a wider support for different I/O standards than microcontrollers, making it possible to select an I/O standard with better signal integrity for the task at hand. More importantly, the programmable logic of FPGAs makes it possible to realize sequential logic circuits capable of toggling outputs once per clock cycle. By comparison, toggling a value in a microcontroller's I/O registers takes multiple clock cycles and may require the execution of multiple instructions. The outputs of an FPGA can be toggled at even higher frequencies by utilizing multiple clock domains and clocking the output circuit at a higher clock than the rest of the FPGA.

For the DDK, we developed a hardware peripheral capable of performing voltage and clock glitching, see Figure 2. This peripheral is capable of performing both voltage and clock glitching as both are supplied to the target device by the channel module. To achieve the high temporal resolution required to induce a glitch for a fraction of a clock cycle of the target device, the output module can be clocked at a higher frequency. To implement this, the A3PN125 used for the DDK provides a PLL, frequency dividers, delay lines and multiple global clock signals. These FPGA hardware resources simplify the process of implementing logic with multiple clock frequencies on the same device.

For our PoC glitcher, a standard smartcard was interfaced via the ISO7816 protocol. For clock glitching, a gated clock output is utilized as the DUT's clock source. The output of the gate depends on the state of both of the inputs. Hence, the glitching module can asynchronously change the output of the gate by altering the `glitch` input of the gate, see Figure 3. By clocking the glitch output module at 100 MHz, 10 ns gltich pulses can be induced in the target device's clock.

Voltage glitching is performed in a similar fashion. Unlike clock glitching, part of the glitching logic must be realized outside of the FPGA. For voltage glitching a PoC circuit was designed that utilized analog multiplexers to select the voltages at v_{cc} and gnd of the target device, see Figure 2. A programmable power supply was used to set the glitch voltage, v_{glitch}. Alternatively a potentiometer or digital-to-analog converter could be used in conjunction with an operational amplifier in a voltage follower configuration. The voltage follower ensures that circuit can drive sufficient current.

The analog multiplexers are controlled by the select lines `s1` and `s2`. The multiplexers switch the supply voltages of the DUT, i.e. v_{cc} and gnd. When `s1` and `s2` are low, the target device received its nominal supply voltage. By setting either `s1` or `s2` to high, v_{glitch} was applied to the high-side or low-side of the device, respectively. The analog multiplexers also included an output enable line, `oe`, that could decouple the supply voltage from the target device completely. As with the clock glitcher, the clock of the glitching module determines the glitch resolution. A 100 MHz clock can produce voltage glitches with 10 ns resolution.

2.3 Software Defined Radio

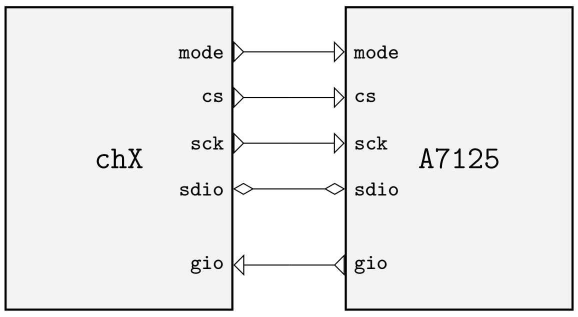

Figure 5: A7125/RF channel module

NRF24-based transceivers, such as the `Nordic`

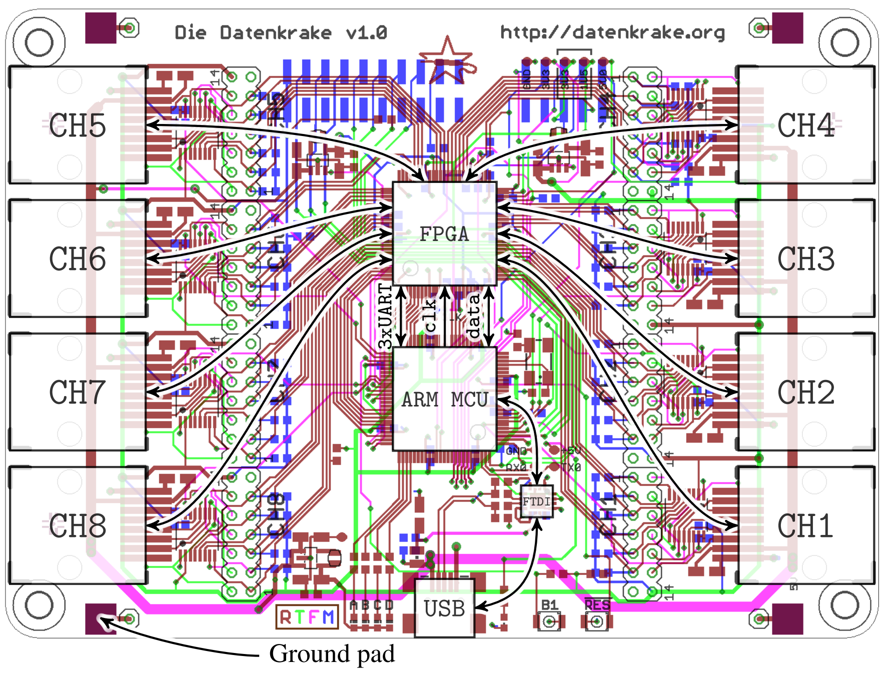

Figure 4: `DDK v1.0` hardware overview.

`Semiconductor NRF24L01+`, are readily available and can be easily acquired. However, these ICs lack the promiscuous mode necessary for raw packet capture. To receive packets, referred to as Enhanced Shockburst Frames, the interface must first be configured with a valid MAC address. Subsequently, incoming packets that match the address configured in the transceiver are decoded and transmitted over an SPI interface. After configuration, outgoing data packets can also be transparently sent to the IC over SPI.

FPGAs are commonly used in software defined radio applications. The nature of programmable logic allows for algorithms to be unrolled and realized as a single clock cycle computation. Even relatively low-speed radio interfaces can overwhelm hardware lacking dedicated logic for the computation. Moreover, variable length frames may make it difficult to identify valid packets in security applications.

The difficulty of implementing such applications is demonstrated by the Keykeriki wireless keyboard sniffer [13]. The Keykeriki is capable of sniffing and injecting the RF traffic of certain transceivers, such as the 2 Mbps NRF24-based modules. Among other applications, NRF24-based modules are widely used in wireless keyboards and car immobilizers. A 2010 presentation demonstrated the vulnerability of such devices to sniffing and injection attacks [14].

To implement a promiscuous mode, a second transceiver is utilzied on the Keykeriki. An `Amicom A7125` 2.4 GHz is used to provide a demodulated bit stream of the wireless interface. The Keykeriki implements an interrupt handler to decode frames during the interrupt routine. The protocol used by Nordic Semi-based transceivers permits packets, known as Enhanced Schockburst frames, that can be up to 329 Bits in length. Furthermore, an Enhanced Shockburst frame can have several variable length fields, see Figure 6. This makes it extremely difficult to detect the start and end of frame without additional information such as the MAC-addresses of the devices that are communicating. Moreover testing all combinations in software is impossible during a single interrupt, since the Keykeriki utilizes just a single 100MHz ARM microcontroller. Hence, the Keykeriki implementation only performs partial real-time protocol analysis and instead takes false positives into account.

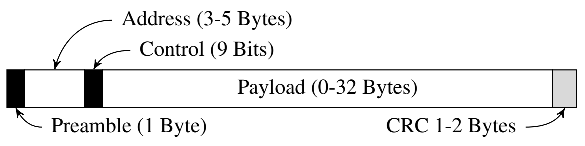

Figure 6: Enhanced Shockburst Frame used by the Nordic Semiconductor RF interface.

A programmable logic implementation would not suffer from these constraints. FPGAs offer several possibilities for simultaneous access to memory by multiple

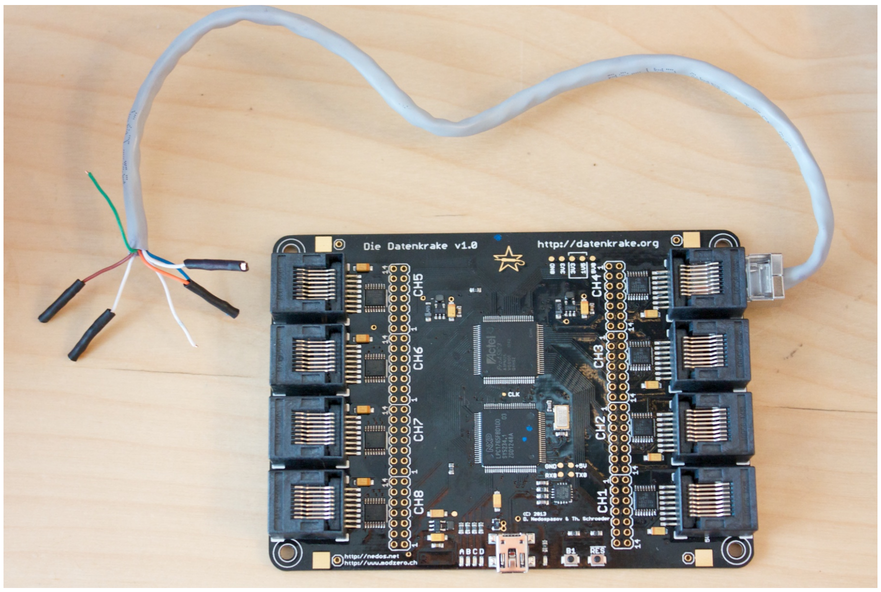

Figure 7: `DDK v1.0` with example test leads.

modules. The A3PN125 used for the DDK includes support for dual ported memory and FIFOs. On the DDK the `Amicom A7125` could be directly connected to one of the channels, see Figure 5. The initial configuration of the A7125 device is performed by using a 3-wire SPI bus (`cs`, `sck` and `sdio`). The `mode` pin is used to put the transceiver module into RX or TX mode. Finally, once configured, the transceiver provides a demodulated signal over `gio`. In such an implementation, the entire signal processing can subsequently be performed in hardware on the DDK. To decode a valid packet, several bits of the frame can be checked using a bitmask. Specifically, the protocol defines two valid preambles: `0xAA` if the first bit of the address is a `1` and `0x55` if the first bit of the address is a `0`. Upon detecting a valid preamble and verifying the control bits, a CRC can be iteratively computed and applied to the rest of the stream. A CRC match would identify a valid packet, which would then be passed to the LPC for additional processing. If dynamic payload length is enabled, the payload length can be used to determine the offset fo the CRC.

3 Design

This section details the characteristics of the DDK's design. An overall overview of the DDK Hardware is presented in Figure 4. One of the primary goals of the DDK project was to provide a simple user-friendly connector that provided access to the GPIO. Test leads should be easy to manufacture and easy interface to PCB test pads, headers and breadboards. The connector should include a power rail in case the target hardware uses a non-standard I/O-interface and a circuit is needed to translate the I/O levels. The I/O should be bidirectional and be capable of interfacing to common embedded open-drain interfaces, such as JTAG and I2C. The firmware of the ARM and the bitstream of the FPGA should be upgradable over the serial interface so that multiple devices can be deployed in parallel and updated.

3.1 DDK Hardware

The DDK consists of an FTDI FT230X USB to serial interface, an NXP LPC 176X ARM Cortex-M3 MCU and a Microsemi ProASIC3 Nano A3PN125 low-cost flash-based FPGA, see Figure 4. To reduce overall system costs and eliminate the need for two separate clock domains, the FPGA clock is supplied via the `CLKOUT` of the LPC, which in turn generates its internal clock from a PLL and a quartz crystal clock source. Because the entire system shares a single clock, this eliminates the potential of clock drift between the FPGA and the ARM. For maximum flexibility, the FGPA and ARM are connected via 6 GPIO pins that form `USART1`, `USART2` and `USART3` of the LPC. With this configuration, custom implementations can easily utilize the LPC's hardware resources to implement additional channels of bidirectional serial communication. Data processed and serialized by the LPC can then be passed to the PC via the FTDI USB to serial interface. Additionally, a 16-bit wide parallel bus connects the LPC and the FPGA. The parallel bus is used to interface the LPC directly to the internal bus of the FPGA.

The board also includes eight configurable I/O channels that are connected to the FPGA. As a readily available connector to interface test leads to external hardware, RJ-45 jacks were chosen. The pinout of the connector consists of the first twisted pair providing a 5 V rail and ground, and the remaining six wires providing user configurable bidirectional I/O. In a pinch, users can cut a standard ethernet cable in half to make a crude set of test leads, see Figure 7. After removing the isolation of the wires, they can be soldered directly to a PCB or inserted directly into a solderless breadboard. In our experience, category 5E spools designed for indoor use are very rigid and optimal for use with solder breadboards. In contrast, ethernet patch cables are significantly more flexible and better adapted for test leads that require crimping and soldering. The 5 V rail makes it easy to realize adapter boards that may need an additional supply voltage for interfacing to other I/O standards or powering additional hardware. Since the USB connector is mini-USB and mini-USB Y-cables are readily available, the DDK can deliver approximately 4.5 W of power to target devices from a connected PC.

To provide additional input protection and ensure the I/O channels can tolerate a wider range of voltages, auto-direction sensing bidirectional voltage translators are utilized to buffer the I/O signals. These chips automatically detect the intended direction of the signal based on the drive strength of either side. Hence, these ICs seamlessly adapt to any I/O configuration of the FPGA without the need to toggle any additional control lines.

Access to the individual inputs and outputs, as well as 3.3 V supply and ground, is available at the headers located along the length of the PCB. I/O signals can be probed directly at the headers, and a ground pad is also included to ground single-ended leads during measurements, see Figure 4. The level shifters also include an output enable control line that can completely decouple them from the circuit. Depending on the interfacing requirements, alternative signal conditioning circuitry can be connected to the headers to support virtually any I/O standard. For example, for open-drain signals lacking on-board pull-up resistors, additional pull-ups can be connected to improve the slew rate of the circuit. Also, if the analog bandwidth of the buffers is insufficient jumpers can short out the inputs and outputs of the level translators creating a direct path from the FPGA outputs to the RJ-45 connectors.

3.2 DDK Architecture

An overview of the DDK architecture is presented in Figure 8. As mentioned in the previous section, the system clock (`clk`) of the FPGA is supplied by the LPC's clock output. Of the 6 UART/GPIO lines interfacing the LPC to the FPGA, only a single UART line (`rx`) is used to

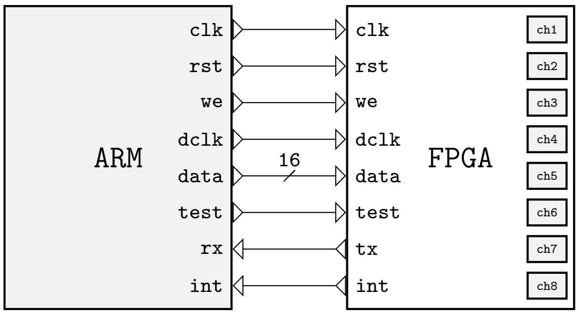

Figure 8: `DDK v1.0` architecture.

send data from the FPGA to the LPC. A dedicated system reset line (`rst`) facilitates synchronous resets of the FPGA. A dedicated I/O interrupt (`int`) alerts the LPC to event triggers implemented on the FPGA. For production testing, a test mode (`test`) was also implemented. The test mode switches the FPGA to a mode of operation in which the integrity of the PCB and all solder joints can be easily verified.

The primary interface for writing from the LPC to the FPGA is the 16-bit parallel bus (`data`). Of the 16 bus lines, 8 are currently used to implement an 8-bit address bus and the remaining 8 are used to implement an 8-bit data bus. Two additional lines are used to provide the write enable (`we`) signal and the data clock (`dclk`), which is toggled upon completion of a bus access. On the FPGA, the open-source Wishbone interconnect is used to interface modules [9]. Data on the parallel bus is internally converted to signals conforming to the Wishbone interconnect on the FPGA. The Wishbone interconnect is particularly common to open-source projects making it easy to integrate existing modules. The overall architecture of the DDK consists of eight I/O channel modules, i.e. `ch1` to `ch8`. The channel modules operate completely independently and in parallel. Users can thus select different implementations for each channel individually depending on their needs, see Section 2.

The ARM primarily configures the FPGA and provides a CLI to the user for configuring the mode of operation. However, the ARM is also connected to the JTAG programming pins of the FPGA and implements a programming interface for upgrading the FPGA bitstream. The ARM itself is upgradable as well, since its boot-ROM contains a serial bootloader. Hence, both programmable devices can be upgraded by users without additional programming hardware.

4 Future Work

The overall design of the current generation DDK is optimized for cost. Future generations of the DDK will be optimized for performance, however they will not replace the current generation DDK.

The current generation DDK utilizes a low-cost Microsemi A3PN125 flash-based FPGA. Though the performance of this device is sufficient for most embedded applications, it lacks several features necessary for advanced embedded analysis. For example, the A3PN125 lacks differential I/O. On the current generation DDK additional I/O standards can be realized by connecting additional circuitry to the channel headers. However, higher-end FPGAs provide a wider selection of I/O standards.

One of the biggest disadvantages of the current generation hardware is the write endurance of the array. Flash-based FPGAs are cheaper to integrate in terms of overall system costs than SRAM FPGAs. The write endurance is not an issue for a low-cost system as systems can simply be replaced. However, a more expensive system should utilize SRAM-based FPGAs that have an unlimited write endurance. A future DDK design would include an SD-Card for storing bitfiles. The bitfiles could then be loaded into the FPGA depending on the task at hand.

For high-speed sampling applications requiring significant trace-depth, the current generation DDK lacks sufficient sample memory. In a future generation a higher performance microcontroller with a dedicated external memory interface will be used. The FPGA will interface the microcontroller to multiple channels of SDRAM. Sample data will simply be mapped into the address space of the microcontroller. To provide a higher-speed interface for data transfers both a high-speed USB interface and an ethernet phy will be integrated into the design. The ethernet phy will also allow devices to operate autonomously and to be configured remotely.

5 Conclusion

In this work we introduced Die Datenkrake. The DDK is a low-cost platform for embedded security analysis, which provides significantly more flexibility than other solutions. The DDK utilizes standard RJ-45 connectors, making it easy to interface to target devices. The programmable logic of the DDK makes it possible to analyze multiple devices in parallel, implement I/O with a very high timing resolution and perform real-time analysis in hardware. The design of the DDK makes it easy to interface existing modules to the internal bus interface. Most importantly the device is completely open source hardware. The design files, as well as the source code are freely available on the project website.

References

[1] Ross J Anderson. Security Engineering. In *A Guide to Building Dependable Distributed Systems*, chapter Physical Tamper Resistance, pages 483–521. Wiley, November 2010.

[2] H Bar-El, H Choukri, D. Naccache, Michael Tunstall, and C Whelan. The Sorcerer's Apprentice Guide to Fault Attacks. In *Proceedings of the IEEE*, pages 370–382, 2006.

[3] Die Datenkrake. Repositories. `http://datenkrake.org/repos`.

[4] Hexrays. IDA: About. `https://www.hex-rays.com/products/ida/index.shtml`.

[5] Andrew "Bunnie" Huang. *Hacking the Xbox: An Introduction to Reverse Engineering*. an introduction to reverse engineering. No Starch Press, 1 edition, July 2003.

[6] Nathan Andrew Fain. JTAG Enumeration. `http://deadhacker.com/2010/02/03/jtag-enumeration/`.

[7] National Instruments. FlexRIO. `http://www.ni.com/flexrio/`.

[8] National Instruments. Virtual Instrument Software Architecture. `http://www.ni.com/visa/`.

[9] OpenCores. Wishbone B4 Specification. `http://cdn.opencores.org/downloads/wbspec_b4.pdf`, 2010.

[10] Peach Fuzzing Platform. Homepage. `http://peachfuzzer.com`.

[11] Dangerous Prototypes. Bus Pirate Homepage. `http://dangerousprototypes.com/docs/Bus_Pirate`.

[12] Radamsa Project. Homepage. `https://www.ee.oulu.fi/research/ouspg/Radamsa`.

[13] Remote Exploit. Keykeriki v2.0. `http://www.remote-exploit.org/articles/Keykeriki_v2_0__8211_2_4ghz/index.html`.

[14] Thorsten Schröder, Max Moser. Practical Exploitation of Modern Wireless Devices: Keykeriki V2. `http://t2.fi/schedule/2010/#speech4`.

[15] I Verbauwhede, D Karaklajic, and J.M. Schmidt. The Fault Attack Jungle-A Classification Model to Guide You. *Fault Diagnosis and Tolerance in Cryptography, FDTC 2011*, pages 3–8, 2011.