FPGA Side-Channel Receivers

Ji Sun
University of Alberta
Edmonton, AB, Canada
ji.sun@ualberta.ca

Ray Bittner
Microsoft Research
Redmond, WA, USA
raybit@microsoft.com

Ken Eguro
Microsoft Research
Redmond, WA, USA
eguro@microsoft.com

ABSTRACT
The popularity of FPGAs is rapidly growing due to the unique advantages that they offer. However, their distinctive features also raise new questions concerning the security and communication capabilities of an FPGA-based hardware platform. In this paper, we explore some of the limits of FPGA side-channel communication. Specifically, we identify a previously unexplored capability that significantly increases both the potential benefits and risks associated with side-channel communication on an FPGA: an in-device receiver. We designed and implemented three new communication mechanisms: speed modulation, timing modulation and pin hijacking. These non-traditional interfacing techniques have the potential to provide reliable communication with an estimated maximum bandwidth of 3.3 bit/sec, 8 Kbits/sec, and 3.4 Mbits/sec, respectively.

Categories and Subject Descriptors
B.4.2 [**Input/Output and Data Communications**]: Input/Output Devices – *channels and controllers*

General Terms
Design, Security.

Keywords
FPGA, side-channel receiver, thermal, phase shift, DDR2, I2C.

1. INTRODUCTION
Prior work on side-channel communication has primarily focused on side-channel attacks [2][4][5][8][9][10]. Side-channel attacks gain information from systems that are assumed to be secure. However, these exploits do not rely on vulnerabilities in the frontline security protocols that are used, but rather on the fact that systems may emanate information via mechanisms that designers may not anticipate. For example, hackers may be able to extract the key used by an encryption chip simply by closely monitoring the power consumption.

That said, centering the discussion on side-channels as an unintentional source of information does not address two important considerations. First, system developers may purposely want to implement side-channel communication. For instance, ICs are often pin-limited and a normally out-of-band communication technique could add valuable I/O capacity [18]. Similarly, such unconventional communication vectors may be able to fix board-level design errors or defects in I/O resources after fabrication and assembly. Side-channels may even be used as a latent signature to watermark a system [3][6].

The second issue that should be considered is the possibility of a side-channel receiver. Although many prior research projects have investigated the information that may *leave* a device, to the best of our knowledge, no prior work has looked at feasible side-channel mechanisms by which information may *enter* a device. Bi-directional communication is not only necessary for side-channels to become generally applicable to the positive (white-hat) uses previously mentioned, we must also consider its impact on the potential negative (black-hat) uses. For example, Trojan covert channels are side-channels created by system developers to victimize end users [1][10]. Although an end user might believe that a system is only transmitting approved information (because they can monitor the in-band communications), the system may also be sending private data via a side-channel. The addition of a side-channel receiver would enable much more sophisticated and stealthy attacks. For instance, transmissions that are conditionally triggered would be far more difficult to detect than those that are either always on or statically triggered.

The mechanisms and implications of side-channel communication are particularly important for the FPGA community. This is because the inherent reprogrammability of FPGAs adds a new opportunity to use, or to fall victim to, a side-channel. For example, although the board-level design of an FPGA-based system may be fixed early in the development cycle, the circuit within the FPGA itself can be updated at any point. Thus, as compared to an ASIC-based system, it is easier to take advantage of the I/O and post-fabrication advantages offered by side-channel communication. On the other hand, reprogrammability also may make FPGAs more vulnerable than ASICs to attack. For instance, a Trojan covert channel does not have to be inserted at the level of the transistor layout, it can be added to a system firmware update.

While FPGAs offer an entry point for side-channel communication late in the development process, they also present their own challenges to actually implementing a functional system. Unlike ASICs, that can be customized to offer a wide range of different and finely-tuned structures (including arbitrary analog circuits), FPGAs have very specific pre-defined resources. These structures either may not be customizable or may only be modified within a given range / in specific increments.

In this paper, we investigate techniques to communicate between an external transmitter and an FPGA-based side-channel receiver.

We present three general mechanisms of communication and demonstrate specific proof-of-concept implementations for each. Highlighted in Section 3, speed modulation techniques use a transmitter that can change the intrinsic operational speed of components inside an FPGA. The focus of Section 4, timing modulation approaches communicate by modifying the delay of an FPGA's I/O signals. In Section 5, we discuss pin hijacking techniques in which we co-opt a wire already in use by an FPGA I/O signal to carry additional side-channel data. We also discuss analytical models for their respective bandwidth to provide some measure of the associated functionality or risk.

2. SIDE-CHANNEL EVALUATION

Although we will demonstrate specific working examples of FPGA-based side-channel receivers, we would also like to get some idea of their more general potential. The viability of any particular side-channel technique will largely hinge upon three factors: the difficulty or cost associated with building the external transmitter, the area required or constraints imposed building the receiver in the FPGA, and the maximum transfer rate that the channel can achieve.

Unfortunately, the cost and convenience of implementing a transmitter or receiver is somewhat subjective and likely depends upon the details of the specific use-case. Thus, while we can report factors such as the resource requirements of our proof-of-concept implementations, the first two considerations that we mentioned above are largely qualitative. On the other hand, we can analyze the data bandwidth potential much more objectively.

As shown in Eq. 1, the maximum transfer rate in bits per second of a given communication mechanism (*B*) is limited by the number of unique symbols we can express (*S*), the sampling rate of the technique (*F*), and the number of independent channels that we can create (*C*). As we describe each of the communication mechanisms in the following sections, we will also discuss how *S*, *F*, and *C* are affected by the specific characteristics of the various approaches.

$$B = log_2(S) * F * C \tag{1}$$

3. SPEED MODULATION

In this section, we focus on communication side-channels in which a transmitter causes changes in the speed of the underlying logic inside an FPGA-based receiver. One mechanism that such approaches can use is that the performance of any integrated circuit is affected by three factors: process, voltage, and temperature (PVT). Although it may be difficult or impossible to externally modulate the technical process parameters and physical structures created on an FPGA during manufacturing, it is relatively easy to manipulate the supply voltage and temperature of the chip.

Intentionally heating or cooling the chip is significant because, in typical operational ranges, there is an inverse relationship between the temperature and propagation delay of a transistor. Increased temperature decreases the mobility of electrons and holes. This results in the reduction of current, which in turn increase RC delay. Similarly, changing the supply voltage of a chip also changes the propagation delay. This is because, in traditional CMOS circuitry, the supply voltage determines the maximum drain-source voltage (V_{ds}). Increasing V_{ds} increases the switching current, which, in turn, reduces RC delay.

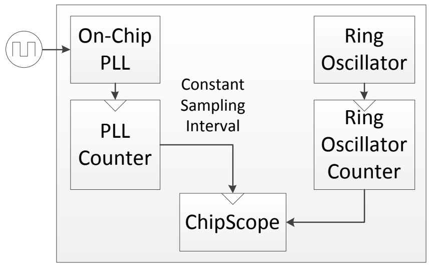

Figure 1: Block diagram of temperature modulation platform

3.1 Case Study – Temperature Modulation

In this paper, we concentrate on temperature modulation. Although theoretically either voltage or temperature could be used for communication, in practice it is likely easier to build a temperature transmitter. This is for two reasons. First, it may be non-trivial to find or build a power system that can actively modulate the voltage provided to the FPGA with sufficient accuracy. Furthermore, such a system would probably require replacing or modifying the existing voltage regulators typically found on FPGA system boards. As will be shown later, the thermal transmitter we built does not require any precision equipment to operate and any modifications made to the FPGA board itself are easily and fully reversible.

To implement a temperature-based side-channel, we borrow a concept from [14] and [17] and use a ring oscillator to detect changes in the temperature of the FPGA. Figure 1 shows a block diagram of our initial testing platform. A ring oscillator provides the clock for a counter (right side of Figure 1). A separate counter, driven from a known frequency clock (left side of Figure 1), is used to sample the ring oscillator counter at a fixed time interval. As the speed of the ring oscillator increases or decreases in response to temperature, the sampled count will correspondingly increase or decrease. These sampled values can be tracked over time to detect changes in temperature.

While the frequency of any given ring oscillator depends upon temperature, how strongly and reliably it reacts to changes in temperature depends upon several factors: the physical characteristics of the FPGA, the number of stages in the oscillator, the relative placement/routing of the LUTs used, and the physical location of the oscillator within the FPGA. As discussed in [14] and [17], these dependencies mean that painstaking chip-specific profiling and calibration is required before ring oscillators can be used as accurate thermometers. However, such calibration is not necessary in our case when temperature is used as a means of communication. This is because we can use changes in temperature rather than absolute values to denote symbols. As will be shown later, this only requires very rough device family-level profiling and calibration.

3.2 Temperature Modulation Bandwidth

As mentioned earlier, the maximum transfer rate that any communication mechanism can achieve is related to the number of communication symbols used (*S*), the sampling frequency (*F*), and the number of independent channels implemented (*C*).

In the case of temperature modulation, S_{Therm} is the number of distinct changes in temperature level used for communication. If we assume that an external transmitter can equally heat or cool the FPGA, we can assign half of the symbols to increases in temperature and half to decreases in temperature. As shown in Eq. 2, the maximum number of usable symbols is defined by the

$$S_{Therm} = T_{Range}/T_{\Delta Min} \tag{2}$$

$$T_{\Delta Min} = (N_{Therm} * \sigma_{Therm})/Slope_{Therm} \tag{3}$$

$$F_{Therm} = \min(F_{\Delta Count}, F_{\Delta Temp}) \tag{4}$$

$$F_{\Delta Count} = N_{Therm} * \sigma_{Therm} \tag{5}$$

$$Voltage(t) = V_{in}\left(1 - e^{-t/RC}\right) \tag{6}$$

$$Temp(t) = (P * R)\left(1 - e^{-t/RC}\right) = T_{\Delta Max} \tag{7}$$

$$F_{\Delta Temp} = \left(-R * C * ln\left(1 - {T_{\Delta Max}}/{P * R}\right)\right)^{-1} \tag{8}$$

operational temperature range that is used (T_{Range}) divided by the minimum change in temperature that the receiver can reliably detect ($T_{\Delta Min}$).[1] As we will show later, the relationship between temperature and ring oscillator speed is highly linear. Thus, the symbols can be evenly spaced within the temperature range used for communication.

As shown in Eq. 3, the minimum discernable temperature change ($T_{\Delta Min}$) is defined by the minimum change in ring oscillator speed we can reliably detect divided by how strongly the ring oscillator responds to changes in temperature. The minimum discernable change in ring oscillator speed is related to the measurement error in the system ($N_{Therm}*\sigma_{Therm}$). This is an arbitrary constant noise margin factor multiplied by the maximum standard deviation in the speed of the ring oscillator when the temperature is held constant at any given point in the operational range, as measured by the sampled ring oscillator counter. $Slope_{Therm}$ is the average change in ring oscillator speed per degree.

As shown in Eq. 4, the maximum sampling frequency (F_{Therm}) is determined by the smaller of two rates. The first term in Eq. 4 ($F_{\Delta Count}$) is the rate at which we can accurately detect changes in temperature – i.e. how long does the ring oscillator counter have to run before we can reliably discern if there was a change in temperature? The second term in Eq. 4 ($F_{\Delta Temp}$) is the rate at which we can change the temperature of the FPGA itself – i.e. considering physical factors, how quickly can heat flow between the transmitter and the packaged FPGA core?

Shown in Eq. 5, the rate at which we can detect temperature changes ($F_{\Delta Count}$) is equal to the minimum change in ring oscillator speed we expect to see between different symbols. That is, if we want to detect an *X* Hz change in the frequency of a ring oscillator, the counter must run for 1/*X* seconds for the difference in the sampled count to be at least one. Thus, we will be able to take a new measurement every 1/*X* seconds, or at a rate of *X* Hz.

The rate at which we can change the temperature of the FPGA ($F_{\Delta Temp}$) is determined by how quickly the transmitter can dissipate heat into or remove heat from the FPGA. As described in [12], the heat transfer between an IC and an attached heatsink can be modeled by making an equivalent "thermal RC circuit". By investigating the impulse response of this circuit, we can predict the temperature response of the FPGA as the thermal transmitter cycles, either heating or cooling.

[1] The accuracy, switching speed, and output quantization of the transmitter will also affect real-world performance. However, for our analysis will assume that these issues will not be the limiting factor, given a well-designed transmitter.

Looking at Eq. 6 and Eq.7, we can see the similarities between the more familiar electrical RC equation and the thermal RC equation, respectively. In Eq.7, *R* is the thermal resistance of the FPGA package and the interface between the thermal transmitter and the FPGA package. *C* is the thermal capacitance of the FPGA and its package. *P* is the power that the transmitter can either generate or absorb. $T_{\Delta Max}$ is the maximum change in temperature corresponding to any symbol used for communication. Rearranging this equation, we can solve for time and determine the minimum time required for the transmitter to overcome the FPGA's thermal inertia and raise or lower the temperature by the maximum necessary amount. Eq. 8 is Eq. 7 solved for the inverse of the time period.

Finally, the maximum number of communication channels (C_{Therm}) equals the number of independent temperature zones we can create on the FPGA. For a variety of practical and mechanical reasons, we will assume that subdividing the FPGA is too difficult and C_{Therm} equals one.

4. TIMING MODULATION

In this section, we focus on side-channel communication mechanisms in which a transmitter overlays its own data on top of an unrelated FPGA I/O signal. This is accomplished by phase shifting the signal or changing its delay. These small differences in signal timing are interpreted by an FPGA-based receiver, separately from the data in the conventional signal that it is piggybacking upon. One potential mechanism that timing modulation side-channels can exploit is a system-level design characteristic found in many devices: timing adaptive or self-aligning I/O. This key feature is included by developers when they cannot be certain, at design time, of the delay of the physical transmission pathway used to carry a signal.

This same general timing issue manifests itself in many different ways. For example, the uncertainty in delay could be at a macro or micro scale. Similarly, the communication could involve either round-trip request/acknowledgements or purely uni-directional communication. Regardless of the specifics, though, all of these situations require two features that also enable side-channel communication. The first characteristic is that designers must use I/O protocols that allow for a range of different signal delays. This creates room for a side-channel transmitter to modulate the delay for communication, on top of whatever data the signal is carrying in the more conventional sense. Second, these I/O protocols must include calibration or alignment capabilities to identify the amount of delay currently present in a signal. This evaluation gives a side-channel receiver a direct way to extract the time-based information.

As a simple example of uncertainty in coarse-grain two-way communication, consider a typical client/server network transfer. Because a client PC cannot predict the network latency for any packet to or from any server, the protocol must keep track of request flight times and allow for a fairly wide time-out window. Any network node between the client and server (or the server itself) could communicate with a latent receiver on the client by purposely adding delay or reordering packets to change the arrival time of the data at the client, beyond the variations that occur in the nominal case. Notice that this modulation can be performed on either outgoing requests or incoming acknowledgments. As long as the total round-trip delay of the data remains within legal bounds, the transfer is otherwise unaffected (we consider introducing errors an entirely different class of communication).

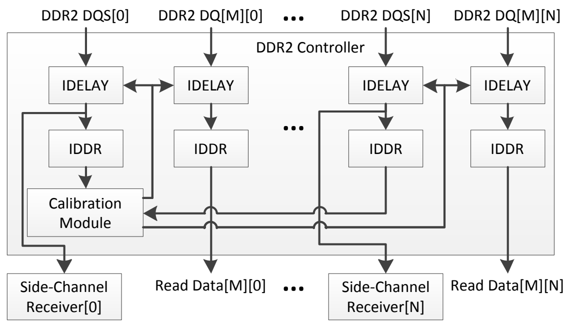

Figure 2: Phase delay detection/re-alignment and phase delay modulation receiver

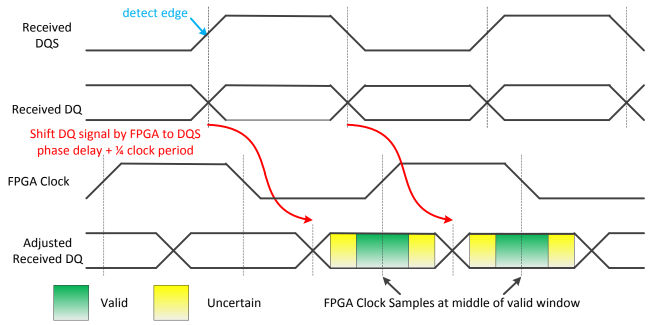
detect edge
Received DQS
Received DQ
Shift DQ signal by FPGA to DQS phase delay + ¼ clock period
FPGA Clock
Adjusted Received DQ
Valid
Uncertain
FPGA Clock Samples at middle of valid window

Figure 3: Aligning DQ using DQS and DQ/DQS misalignment

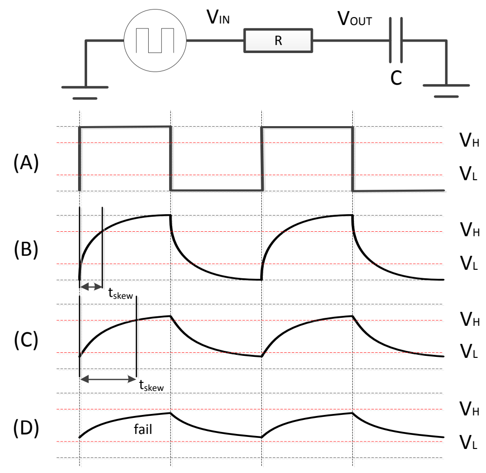

Figure 4: Slew rate with respect to trace capacitance

4.1 Case Study – Phase Delay Modulation

In this paper, we explore the possibility of fine-grain timing modulation implemented on high speed communication lines. Specifically, we add a delay-based signal to a DDR2 memory interface. We exploit a feature built into the interface that is intended to allow circuit designers to compensate for minor differences in timing due to board-level constraints, manufacturing variability, or unpredictable factors that change over time.

A typical DDR2 memory interface consists of clock, control, address and data lines. Precisely controlling the absolute and relative timing of these signals is critical due to the high switching rates that are used. As explained in [7], the data (DQ) bus of a DDR2 module is separated into several smaller logical buses. Each of these buses also typically includes its own data strobe (DQS) line. Although the entire data bus theoretically operates off of a common clock, this global clock is only used for coarse-grain synchronization. The precise alignment of the data in each of these smaller buses is performed with respect to its own DQS signal. This smaller-scale DQ/DQS bundling is implemented to ease a variety design problems. For example, only the board-level traces within each sub-bus need to be path-length matched, rather than all of the traces for the entire data bus.

The lack of global bus synchronization means that when the FPGA receives incoming data from the DDR2 module, the data in each of the smaller DQ/DQS buses is potentially out of phase with respect to all of the others. Thus, each of these buses needs to be individually re-aligned so that the data can be sampled by a single local clock on the FPGA. As seen in Figure 2, the open-source memory controller [7] we used in our proof-of-concept system has on-board calibration and delay circuitry to perform this re-alignment. As seen in Figure 3, the controller determines the current phase delay on each DQ/DQS bus with respect to the FPGA clock by locating the rising edge of its DQS signal. As seen in Figure 2, the DQ signals are sent through IDELAY blocks (fine-grain delay elements), so the calibration module can arbitrarily phase shift the input data by changing the setting of the IDELAY blocks. As seen in Figure 3, the controller sets the IDELAY blocks so that the FPGA will sample the input data near the center of its valid window.

One key characteristic of this alignment process is that, even during normal operation, the controller must periodically re-evaluate the phase delay of the incoming DQ/DQS buses and adjust the IDELAY settings to maintain proper alignment with the FPGA clock. This dynamic adjustment is required to handle factors that might change the signal timing during runtime, such as temperature (as discussed in Section 3).

An external side-channel transmitter can also impose its own influence on the timing of I/O signals. For example, Figure 4 shows an RC circuit. Increasing the capacitance on the output node increases the slew rate of the output signal. Looking at the system from a digital standpoint, this delays the point at which the output crosses V_H, the threshold voltage for an input logic “1”. The same holds true for the transition to V_L, the threshold voltage for an input logic “0”. Thus, when we add capacitance to a wire (at least up to some limit), it is perceived by any digital circuit that uses this signal as an input as an increase in timing skew.

Applying this concept to the connection between an FPGA and a DDR2 module, a phase delay transmitter could modulate the capacitance on the board traces used by a DQ/DQS bus. As shown in Figure 2, these changes in capacitance could be detected within the FPGA by monitoring the IDELAY settings. As long as the added capacitance does not overwhelm the driver on the memory module, the conventional data transfer between the memory module and the FPGA would not be interrupted.

4.2 Phase Delay Modulation Bandwidth

When we look at the factors that determine the achievable bandwidth of a phase delay modulation communication channel, we can see parallels to our earlier analysis of temperature modulation channels. For example, Eq. 9 is very similar to Eq. 2 – i.e. the number of symbols that can be used (S_{Phase}) is defined by the range of capacitance that can be added to an existing I/O trace by the transmitter (C_{Range}) divided by the minimum capacitance that can be measured by the receiver, in the form of phase delay

$$S_{Phase} = C_{range} / C_{\Delta min} \quad (9)$$

$$C = \frac{-t}{\mathrm{R}*\ln\left(1 - {}^{V(t)}/_{Vin}\right)} \quad (10)$$

$$C_{max} = C_{range} + C_{Para} = \frac{-t_{\Delta max}}{R_{Drive}*\ln\left(1 - {}^{(V_H - V_L)}/_{V_{dd}}\right)} \quad (11)$$

$$t_{\Delta max} = \min\left(t_{1/_2 signal}, \ t_{\Delta Thres}\right) \quad (12)$$

$$C_{\Delta min} = \frac{-(t_{\Delta min})}{R_{Drive}*\ln\left(1 - {}^{(V_H - V_L)}/_{V_{dd}}\right)} \quad (13)$$

$$t_{\Delta min} = t_{minQuant} * \lceil N_{Phase} * \sigma_{Phase} \rceil \quad (14)$$

($C_{\Delta min}$). However, unlike temperature modulation, phase delay communication must piggyback upon an existing I/O signal. The characteristics of this signal heavily affect the transfer rate that can be achieved.

The range of capacitance that can be used for phase delay communication (C_{Range}) is determined by several factors: the frequency of the signal it is overlaid on and the physical attributes of the signal driver, the board-level trace, and the FPGA's input pad. If we model the connection between the driver on the DDR2 module and the input pin of the FPGA as a simple RC circuit, we can illustrate the relationship between these terms.

Eq. 10 is the basic voltage equation for an RC circuit (Eq. 6), solved for capacitance. If we substitute a few variables into Eq. 10, we get Eq. 11. In this case, the maximum load capacitance that the memory module can drive without causing errors is C_{Max}, or the sum of the maximum capacitance that can be added by the transmitter (C_{Range}) and the inherent parasitic capacitance of the connection itself (C_{Para}). C_{Max} is dependent upon the maximum timing skew that can be introduced ($t_{\Delta max}$), the drive voltage of the memory (V_{dd}), the voltage gap between a logic "1" and a logic "0" on the FPGA's input pin (V_H - V_L), and the effective resistance of the memory module driver (R_{Drive}).

As shown in Eq. 12, the maximum allowable timing skew ($t_{\Delta max}$) is the minimum of two periods. The first factor is ½ the period of the signal we are piggybacking upon ($t_{1/2signal}$). That is, if the input pad on the FPGA does not need any setup time, the input signal can reach V_H just as the driver is switching from a logic "1" to a logic "0". On the other hand, the skew may be limited by some internal threshold built into the calibration logic within the communication controller ($t_{\Delta Thres}$). For example, the DDR2 controller that we use in our proof-of-concept system does sanity-checking after the system is initially calibrated, somewhat limiting the maximum allowable skew after this point.

Looking at Eq. 13, we can see that the minimum change in capacitance that can be detected by the receiver ($C_{\Delta min}$) is dependent upon (V_H - V_L), V_{dd}, and R, the same as C_{range}. However, it is also dependent upon the minimum phase delay that can be accurately measured ($t_{\Delta min}$). As shown in Eq. 14, $t_{\Delta min}$ is the minimum time quantum that the calibration system can implement ($t_{minQuant}$) multiplied by an arbitrary constant noise margin factor (N_{Phase}) times the maximum standard deviation in the phase delay when the capacitive load is held constant, as measured by the calibration circuitry (σ_{Phase}). In most systems, $t_{minQyuantum}$ will be defined by the timing granularity of the I/O delay elements. In the case of our proof-of-concept system, this is the minimum timestep of the IDELAY blocks. ($N_{Phase}*\sigma_{Phase}$) must be rounded up to the nearest integer because, by definition, we cannot detect or implement fractional $t_{minQyuantum}$.

Completing our bandwidth analysis, the maximum sampling frequency (F_{Phase}) is equal to the rate at which recalibration is performed by the receiver. The maximum number of communication channels (C_{Phase}) is equal to the number of independently calibrated signals.

5. PIN HIJACKING

In this section we focus on side-channel communication mechanisms in which a transmitter inserts its own data onto an I/O wire that has already being used by another signal. These communication techniques rely on the fact that there is often "dead time" in a signal or on a bus – i.e. time when the connection is not actively being used for communication. A pin hijacking side-channel transmitter can listen for this idle time (or otherwise know when it will occur) and transmit its own data during this interval. The side-channel can detect and capture the incoming data on the FPGA with its own receiver. As with the timing modulation side-channels, this general concept can be applied on a macro or micro scale.

As a simple example of micro-scale "idle time", consider a typical connection between an external data source and a receiver inside an FPGA. The source and destination must be synchronized, so the FPGA-based receiver might be designed to capture the input data on the rising edge of some common clock. This signal likely wastes at least some portion of the intrinsic bandwidth of the underlying connection. We say this because, unless the signal is running near the limit of the effective setup and hold times of the receiver, the receiver is "ignoring" the input for some portion of the clock period. If the source driver and board-level traces had sufficient headroom to do so, the transmitter could send data on both the rising and falling edges of the clock. Essentially, we can transmit in a double-data rate manner, but time-multiplex between the data from the original signal and data from an entirely new signal. The original receiver would not perceive any difference, but we could add a new receiver to the FPGA that samples on the falling edge of the clock to capture the side-channel data. This same general double-data rate concept can be extended to an *N*-way data rate connection, only limited by the signaling capability of the physical connection.

5.1 Case Study – I^{2}C Interface Hijacking

In this paper, we looked at a bus that displays very coarse-grain segments of idle time. We built a pin hijacking communication side-channel on top of the dedicated I^{2}C bus that connects the DDR2 memory controller on the FPGA with the Serial Presence Detect (SPD) chip on the memory module itself. In addition to simply demonstrating the feasibility of a pin hijacking side-channel, it also shows that such communication techniques do not require pure input data lines, but can be built from bi-directional pins as well.

DDR2 memories generally contain an SPD EEPROM. This chip stores timing information for the memory, including the maximum clock rate, CAS latency, required refresh rate, etc. The SPD data is normally only accessed when the system is booting so that the FPGA can properly configure the memory controller. After this point, the I^{2}C connection is typically left idle.

The I^{2}C protocol was first introduced by NXP [11]. The simplest I^{2}C structure consists of a master node, a slave node and two open collector, bidirectional signals: the Serial Clock (SCL) line and the Serial Data (SDA) line. The master node first initiates communication with the slave by actively driving the SCL and

SDA lines. It transmits control and address information, then relinquishes control of the SDA line. The slave node listens for this data and, if it is responsible for the provided address, sends back an acknowledgement, actively driving SDA. This establishes a connection between the master and slave until a stop signal is sent. After the transfer is complete, the SCL and SDA lines return to a floating state. Figure 5a shows the normal connection between the memory controller and the SPD module.

Our side-channel connects an additional I^2C master to the SDA and SCL lines and adds a small amount of logic to the memory controller that makes it first act as an I^2C master (to obtain the timing information from the SPD module) and then as an I^2C slave (to obtain data from the side-channel transmitter). Figure 5b shows the modified system. The side-channel transmitter merely waits for the memory controller to complete its transaction with the SPD module and go dormant. After this point, the side-channel transmitter can re-activate the bus and create a new connection between itself and the side-channel receiver.

This same mechanism can be implemented on any standard interface that uses a high impedance state. For example, the data pins of the DDR2 interface are generally tri-stated during normal operation. During that time, it would be possible to set up a side-channel with a very wide bus (64-bits or more) running at high speed. As long as the side-channel has some sort of negotiation so that it can relinquish control of the bus when the rightful user wishes to read or write to the DDR2 memory, it will not impede normal operation. Naturally, as the complexity of this sort of side-channel increases, so does the logic and power requirements for its implementation.

5.2 Pin Hijacking Bandwidth

As with any other digital signal, the number of symbols ($S_{PinHijack}$) and channels ($C_{PinHijack}$) of a pin-hijacking side-channel is determined by the characteristics of the wire or bus that it is based upon. Similarly, so is the sampling rate. However, unique to side-channel communication, we must take into account the fraction of time that the side-channel can actually communicate. As seen in Eq. 15, the effective sampling frequency ($F_{PinHijack}$) is dictated by the maximum allowable clock rate of the basic channel ($F_{Channel}$) multiplied by the fraction of time that the channel is typically unused (*FracIdle*).

$$F_{PinHijack} = F_{channel} * FracIdle \qquad (15)$$

6. Implementation and Results

To demonstrate the feasibility of the three communication mechanisms introduced in this paper, we built proof-of-concept implementations. All of the side-channel receivers were built on a Xilinx Virtex-5 XUPV5-LX110T prototype board. All of the transmitters were built from materials easily obtained from an electronics parts supplier for less than roughly 100 USD.

6.1 Temperature Modulation

Our experimental thermal communication system is shown in Figure 6. The external transmitter consists of a Peltier thermoelectric device, a standard benchtop power supply, and a water-cooled heatsink. When a Peltier device is driven by a DC current, it actively pumps heat from one side of the device to the other. When the Peltier device is cooling the FPGA, the amount of heat that it removes is proportional to the current provided by the power supply and how efficiently the heat is taken away from the other side of the device. The water-cooled heatsink provides a

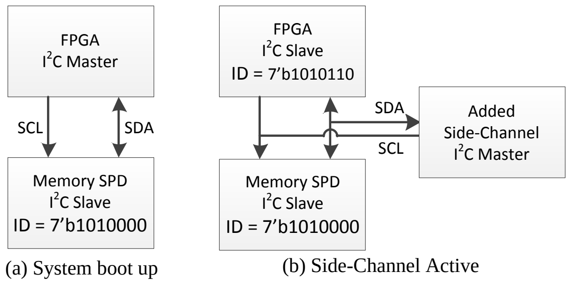

Figure 5: SPD I^2C connectivity

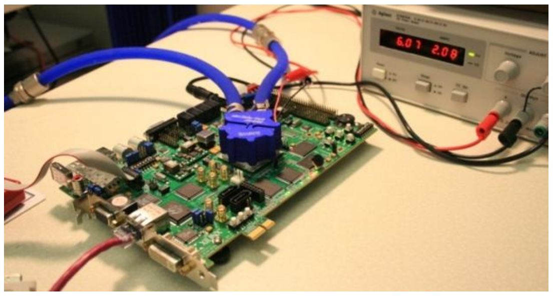

Figure 6: Temperature modulation platform

Table 1: **Experimentation variables**

# of stages	2, 3, 4, 5, 6, 8, 10, 14 and 20
Physical Location	25 uniformly spaced locations
Temperature	-15 to 85°C, in 10° intervals

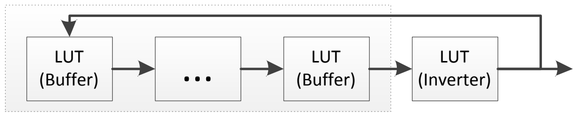

Figure 7: N-Stage Ring Oscillator

thermal mass into which the Peltier device can dump heat. Peltier devices are sensitive to polarity and the transmitter can be switched from cooling the FPGA to heating it simply by reversing the power connection. In this case, the mass of the heatsink provides a source of heat that can be driven into the FPGA. Although we used a water-cooled heatsink because it provided a very large thermal mass so that we could conduct extended heating and cooling experiments, a more conventional heatsink would likely be sufficient for a typical thermal transmitter.

Based on our discussion in Section 3.2, how reliably we can measure the speed of a ring oscillator and how strongly it responds to changes in temperature play a key role in determining the achievable bandwidth of the system. Thus, we performed a series of experiments to determine how the structure and location of the ring oscillator affects these considerations.

As seen in Table 1, we varied the number of stages in the ring oscillator from 2 to 20. As seen in Figure 7, the number of stages in the ring oscillator can be varied arbitrarily by using one inverter and inserting (N-1) buffers. We also looked at the effect that location had on the ring oscillators. To ensure the largest degree of consistency and the fastest operational speed, we hand-placed the ring logic as densely as possible and positioned the associated counter in an adjacent column. Each CLB in the Virtex-5 contains two slices, each slice contains four LUTs, and each of the LUTs can implement two independent one-input functions. Thus, all of the ring oscillators that we looked at could fit into a

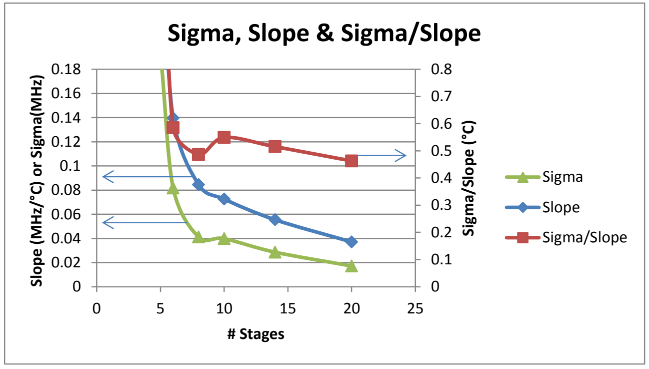

Figure 8: Standard deviation, thermal response and standard deviation/thermal response as a function of the # of ring oscillator stages

Table 2: Characteristics of top 3 ring oscillator candidates

# Stages	20	10	8
Avg. RO Freq @ 25°C (MHz)	62.06	119.9	142.0
σ_{Therm} (MHz)	0.46	0.55	0.49
Avg. $Slope_{Therm}$ (MHz/°C)	0.037	0.073	0.085
$T_{\Delta Min}$ (°C for N_{Therm}=2)	0.93	1.09	0.97
$F_{\Delta Count}$ (KHz for N_{Therm}=2)	34.3	79.6	82.2
$F_{\Delta Temp}$ (Hz for N_{Therm}=2)	3.48	2.89	3.31
B_{Therm} (bits/sec)	3.48	2.89	3.31

Table 3: Resource requirements of thermal receiver

Slices*	117	0.6% of LX110T
6-Input LUTs*	295	0.4% of LX110T
Registers	246	0.4% of LX110T
PLL	1	16% of LX110T (assuming an independent PLL is used)

**Assuming an 8-stage ring oscillator*

maximum of two CLBs, not including the counter. To determine if there was any regional variation in performance across the chip (either due to manufacturing variation or a possible temperature gradient), we implemented 25 independent ring oscillator/counter pairs spread uniformly across the die in a 5x5 pattern. The components within each of these instrumentation rigs had the same relative placement and routing.

As seen in Figure 1, the duration that the ring oscillator counter runs between samples is determined by the reference clock frequency of the onboard PLL and the constant value used by the reference counter to trigger the sampling logic. We used a 200 MHz reference clock sampling every 65,535 (0xFFFF) clock cycles in our testing. We measured each different type of ring oscillator at each of the 25 locations 4,096 times at temperatures between -15°C and 85°C.

Even when comparing very preliminary ring oscillator testing results with an estimate based upon reasonable values for R, C, and P[2], it is immediately clear that the sampling rate of the system is heavily dominated by how quickly we can change the temperature of the FPGA ($F_{\Delta Temp}$). As we will discuss later, while the ring oscillators can detect temperature changes on the order of 1°C at a sampling rate on the order of kilohertz, the Peltier device requires hundreds of milliseconds to transfer enough heat into or out of the FPGA to discernibly change the temperature. It is unlikely that the "drive capability" of the thermal transmitter will change drastically across smaller and larger devices. This is because both the power that can be absorbed or emitted by the Peltier device and the thermal capacitance of the FPGA plus packaging are roughly proportional to the surface area of the chip. Similarly, the thermal resistance of the FPGA/Peltier interface is roughly inversely proportional to the surface area of the chip. Thus, looking at Eq. 8, the ($R*C$) and ($P*R$) terms will largely remain constant regardless of the size of the FPGA package.

Since we now know that F_{Therm} will be determined by $F_{\Delta Temp}$, we can consider the relationship between the number of symbols ($log_2(S_{Therm})$) we have and the sampling frequency – i.e. we can try to maximize ($log_2(S_{Therm})$ * F_{Therm}). While $log_2(S_{Therm})$ grows linearly when T_{Range} is doubled (assuming that $T_{\Delta Min}$ is fixed for a given ring oscillator), according to Eq. 8 $F_{\Delta Temp}$ shrinks super-linearly when $T_{\Delta Max}$ is doubled. Thus, the bandwidth of the system is highest when we minimize the maximum change in temperature used for communication. This means that $T_{\Delta Max}$ should equal $T_{\Delta Min}$. By extension, this means that we should only use two symbols for communication (+/-$T_{\Delta Min}$) and we should prioritize ring oscillators with the smallest variability and the largest thermal response (minimize $\sigma_{Therm}/Slope_{Therm}$).

In Figure 8, we graph the results of our thermal modulation experiments. The vertical axis represents either the maximum standard deviation of the ring oscillator speed across all 11 temperatures and all 25 locations, the average thermal response (slope) across all locations, or the maximum standard deviation divided by the average slope, respective to the corresponding green, blue and red lines. The horizontal axis represents the various ring topologies in increasing order of the number of stages. Table 2 shows the various characteristics of the three ring oscillators with the smallest $T_{\Delta Min}$. We use these experimentally determined $T_{\Delta Min}$ values to calculate $F_{\Delta Temp}$ and, assuming that S_{Therm} and C_{Therm} both equal one, the potential achievable bandwidth of a communication channel that uses one of these rings (B_{Therm}).

Looking at these results, we can draw several conclusions. First, the variability in the speed ring oscillators (σ_{Therm}) decreases as the size of the ring oscillator grows. This is likely because more stages allows the random jitter that is present in the delay of each individual stage to be averaged, creating a more uniform period overall. Second, the thermal response ($Slope_{Therm}$) also decreases as a function of the number of stages. This is somewhat surprising, although it may be caused by the fact that the delay along larger rings contains a greater fraction of interconnect to logic delay – i.e. the interconnect resources may be far less sensitive to temperature as compared to the LUTs. No matter the cause of this phenomenon, since both σ_{Therm} and $Slope_{Therm}$ decrease as a function of the number of oscillator stages, $T_{\Delta Min}$ is roughly constant when more than about eight stages are used. A ring oscillator with eight stages is particularly attractive because it can be implemented entirely within a single Virtex-5 slice. Since this represents the best combination of size and minimum $T_{\Delta Min}$, we use this ring to report the resource requirements shown in Table 3.

Lastly, unlike the FPGA-based thermometer systems in [14] and [17], none of the equations from Eq. 2 to Eq. 8 contain a reference to the absolute intrinsic operating frequency of the ring oscillator.

[2] From the information in [15] and [16], we estimate R, C, and P as 0.13 °C/W, 8.13 J/°C, 30 W, respectively. The thermal capacitance is calculated assuming that the specific heat and density of the IC package are roughly equal to that of solid Al.

We are only concerned with the change in frequency as a function of temperature, or $Slope_{Therm}$. If we combine this with the fact that we only use symbols represented by +/-$T_{\Delta Min}$ (and thus can use conservative positive and negative thresholds), we eliminate the need for chip-specific calibration. Although larger noise margins may affect the achievable bandwidth to a certain degree, it is likely that very approximate testing can determine a reasonable range for $Slope_{Therm}$ that can be used across an entire device family or possibly even all chips made with a given fabrication process.

6.2 Phase Modulation

The XUPV5 board has one DDR2 SO-DIMM socket with eight DQ/DQS sub-buses. Every sub-bus contains one DQS signal that provides synchronization across eight DQ data signals. Our proof-of-concept phase delay transmitter only adds capacitance to the DQS lines rather than all the signals in the DQ/DQS buses. This still creates perceived phase shift across the entire bus because, as discussed in Section 4, the calibration mechanism in the memory controller only examines the DQS signal when determining the phase delay of the bus. We only modulate the capacitance on the DQS signals because it greatly simplifies the physical implementation of the transmitter. Although adding delay unsymmetrically alters the strobe/data signal alignment, we found that we could add almost ¼ of a clock period of phase delay without causing read errors. This is because, as shown in Figure 3, the effective valid window for the data driven by the DDR2 module is fairly wide, even when the memory interface is running at 200MHz (DDR2-800).

Each DQS strobe is actually a differential signal that provides more reliable high-speed communication: DQS and its logical complement, DQS#. As shown in Figure 9, there are two different capacitor topologies that can be used to add delay to the strobe signal. On the left we show a "differential-mode" connection with two capacitors separately connecting DQS and DQS# to ground. On the right we show a "common-mode" connection with one capacitor connected between the drivers. In our early testing, we found that a single capacitor connected between the differential lines provided a much more significant loading effect as compared to the dual capacitor arrangement. This phenomenon is due to the fact that the common mode connection results in two active drivers pulling in opposite directions at all times, essentially doubling the change in voltage that the capacitor experiences during each transition.

The "common-mode" connection topology also had an important advantage over the "differential-mode" technique – it simplified the construction of the phase delay transmitter because we did not have to build a ground plane and did not have to make as many connections to the SO-DIMM. As we will discuss later, simplifying the structure of the transmitter greatly reduced the amount of error introduced into the system.

Figure 10 shows our experimental phase modulation communication setup. We built a phase delay transmitter by mounting a small perfboard on top of the DDR2 SO-DIMM on our FPGA board. This allowed us to attach sockets to the DQS and DQS# pins of the memory module. The red circles in Figure 10 highlight the small wires soldered between the SO-DIMM and the perf-board. We used these sockets to modulate the capacitive load on the strobe lines by manually swapping in and out discrete ceramic capacitors with different values. As mentioned earlier, the effect of the attaching these various capacitors was measured

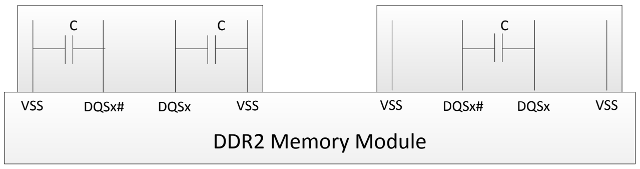

Figure 9: Two capacitor topologies

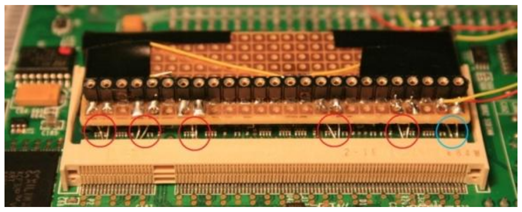

Figure 10: Phase modulation platform

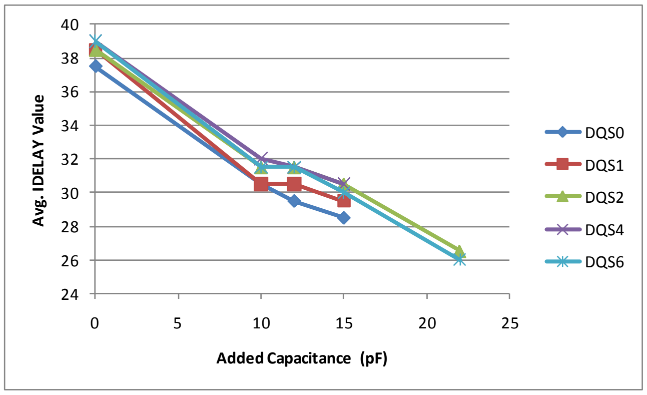

Figure 11: Phase shift as a function of capacitance

Table 4: Characteristics of phase modulation prototype

σ_{Phase}	0.5771 IDELAY units (45.1 ps)
$t_{\Delta Min}$ (N_{Phase}=2)	3 IDELAY units (234.4 ps)
$t_{\Delta Max}$*	12 IDELAY units (937.5 ps)
S_{Phase}	2
F_{Phase}	1KHz

Assuming that the system is not limited by the accuracy of the transmitter
**Determined graphically*

Table 5: Resource requirements of phase modulation receiver

Slices	68	0.4% of LX110T
6-Input LUTs	72	0.1% of LX110T
Registers	132	0.2% of LX110T

by looking at the change in IDELAY settings from the calibration circuitry in the FPGA.

The results of our testing are shown in Figure 11 and Table 4. The phase shift detected by the calibration module was measured 1,000 times on each DQS signal for each capacitance value. We only experimented using five of the eight DQS strobes (0, 1, 2, 4, and 6) on the DDR2 memory. This is because these signals were simply more readily accessible. The memory slot on the XUPV5 board is mounted parallel to the system board and the I/O pins for these strobe signals were located on the exposed side of the SO-DIMM rather than the side facing the main PCB.

Looking at Figure 11, we see that the phase shift increases approximately linearly as a function of the applied capacitance (notice, as the memory controller perceives more intrinsic phase delay on the strobe line, it shifts the data less using the IDELAY blocks to achieve the same total phase delay). This linear relationship is predicted by Eq. 10, if we solve for time rather than capacitance. This confirmation gives us confidence that we can use the data from our experiments to determine relative trends between capacitance and phase shift. That said, we did not apply this data directly to Eq. 9, Eq. 11, and Eq. 13 (the equations that involve specific values for the capacitance that is added to a signal). Our uncertainty regarding plugging our experimental data directly into these equations is caused by the fact that there is a large and somewhat unpredictable difference between the capacitance that is physically applied to a wire in our experimental setup and the capacitance that is actually perceived by the DDR2 module and the FPGA.

The effective capacitive load on the DDR2 drivers is heavily affected by the length of the wires used to connect the capacitors to the board-level traces. This is because, due to the high clock frequency of the DDR2 interface, wire inductance is a significant concern. In our initial testing, we found that leads as short as 0.75” created enough inductance to completely isolate the capacitors from the board traces. That is, no matter how large a capacitor we applied, there was no significant change in the signal phase. In our prototype transmitter design, the wires between the SO-DIMM and the perf-board need to be at least 0.25” for mechanical reasons. Furthermore, due to the fact it was constructed by hand, there is visible variation in the length of these connections. Put together, these factors make it very difficult to determine exactly how much capacitance is actually perceived by the memory module and the FPGA. A full implementation of a phase modulation communication system would need to address inductance. Precise capacitors, a fully electronic switching network, and a more elegant method of attachment would be helpful to achieve consistent communication and maximum bandwidth.

Despite our uncertainty regarding the exact applied capacitance, we can still use the results from our experiments to gain insight into the bandwidth potential of this approach – at the very least, this system represents a lower bound. As seen in Table 4, the maximum standard deviation in the phase delay measured by the memory controller (σ_{Phase}) across all of our experiments was 0.58 IDELAY units (one IDELAY unit is $t_{minQuantum}$, or 78.125ps). If we assume N_{Phase} equals three, by Eq. 14 the minimum phase delay that can be accurately measured ($t_{\Delta min}$) equals ($3 * t_{minQuantum}$) or 234.4ps. Thus, looking at Figure 11, we are able to clearly distinguish at least two symbols: applying or not applying a 10pF capacitor. Given two symbols of data per DQ/DQS bus and eight pairs of DQS lines[3] in the test system, ($log_2(S_{Phase}) * C_{Phase}$) equals 8 bits. F_{Phase} is equal to the recalibration rate of the memory controller. The default setting in the memory controller from [7] is 1 KHz. Thus, the maximum bandwidth into the chip would be 8 Kbits per second. Table 5 shows the resource requirements of the phase modulation receiver.

6.3 Pin Hijacking

Our proof-of concept implementation of a pin hijacking side-channel is shown in Figure 12. As discussed earlier, a conventional I^2C slave is on the SPD chip on the DDR2 module. The dual-mode I^2C master-turned-slave for our side-channel receiver is instantiated in the memory controller on the FPGA.

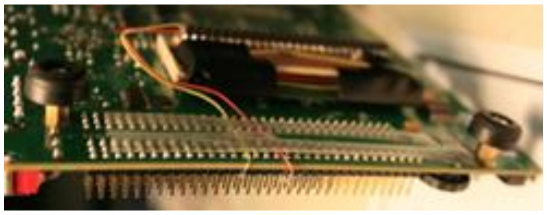

Figure 12: Pin hijacking platform

Table 6: **Resource requirements of pin hijacking receiver**

Slices	90	0.5% of LX110T
6-Input LUTs	162	0.2% of LX110T
Registers	86	0.1% of LX110T

These two modules communicate via the onboard traces of the XUPV5 board.

Although the side-channel transmitter I^2C master would normally be implemented on a separate device, for simplicity sake we built it on the same FPGA as the rest of the system. The I/O signals for the side-channel transmitter are fed through two GPIO pins of the XUPV5 board. These are the red and yellow wires in Figure 12 connecting the GPIO pins on the bottom side of the board with the DDR2 module board traces on the top side. We tapped into the I^2C signals of the DDR2 module using the same perfboard and socket technique described in Section 6.2. These connections are highlighted with the blue circle in Figure 10.

The number of symbols ($S_{PinHijack}$) for an I^2C connection is two – it is inherently binary since there is only one data wire, SDA. As per the official specification of I^2C, $F_{Channel}$ is between 100 KHz and 3.4 MHz, although modern devices are generally capable of communicating at a faster rate. If the FPGA only queries the SPD timing data when the system is booted, the active time of this connection is amortized to zero. Thus, *FracIdle* is nearly one. Lastly, since we only have one I^2C bus, the number of channels ($C_{PinHijack}$) is one. Thus, the maximum achievable bandwidth of our side-channel ($B_{PinHijack}$) equals 3.4 Mbits per second. The resource requirements for our pin hijacking side-channel receiver are shown in Table 6.

Overall, the structure of the I^2C interface helps us implement this type of signal hijacking in two ways. First, the protocol has a built-in notion of addressing, so we can send data between the side-channel transmitter and receiver without activating other I^2C slaves that might be attached to the bus. Second, the protocol explicitly shuts out other I^2C devices once a connection has been established, until the stop signal is sent. This gives us the ability to transfer an infinite-sized payload without violating the bus protocol.

7. Future Work

Although we have demonstrated the feasibility of using speed modulation, timing modulation and pin-hijacking to communicate with an FPGA, there are still many issues that we would like to investigate looking forward.

First, our prototype implementations showed that these communication mechanisms can transmit and receive a coherent signal. However, we have not addressed the issues that surround how we might encode real data. For example, our thermal modulation uses two symbols for communication, an increase or decrease in temperature. However, we cannot use direct binary translation to encode data, because this cannot handle an input

[3] A DDR2 DIMM (rather than SO-DIMM) has 16 DQ/DQS buses. DIMMs with ECC add an additional two channels.

data stream with an unbalanced number of zeros and ones. A straightforward technique to avoid runaway temperatures would be a return-to-zero encoding scheme, but might there be a more efficient solution?

We would also like to look at how these communication techniques might interact with each other. On one hand, different methods of information transfer might interact negatively with one another. For example, variations in temperature might change the real-world noise margins for a communication channel that uses phase modulation. On the other hand, combining different approaches into a hybrid mechanism may significantly increase the achievable bandwidth and/or the difficulty in detecting that a side-channel receiver is present.

Along the same lines, we touched on the general concept of noise and how it could be introduced into the system by any number of factors – even simple things like the length of certain wires might be important. Thus, communication errors are bound to occur. We would like to look at what kinds of error correction might best suit our side-channel communication mechanisms.

Furthermore, much more extensive testing is required to determine the real-world operational limitations of these side-channel mechanisms. For example, our thermal receiver was not tested alongside a real working circuit. While we assumed that the heat created by most host circuits would be relatively constant on the time-scale used by our thermal communication system (and thus would be a "DC" component to the temperature modulation), is there anything that can be done to lower the potential influence of the rest of the system? Could an active feedback thermal transmitter help?

Lastly, in this paper we considered the difficulties in creating side-channels. However, we should also look at how we might break them. For example, since side-channels might be used for nefarious purposes such as leaking private information, what countermeasures might be deployed to protect users? On the other hand, if we use these techniques to watermark our IP, how might pirates try to obfuscate these identifiers?

8. Conclusions

It is unclear whether side-channel communication represents a benefit or a liability to system developers and end users. While side-channels have been classically profiled as a security risk, they have the potential to add new unique capabilities to critical FPGA concerns. Either way, though, the increasing popularity and the inherent flexibility of FPGAs puts them at the forefront of this discussion. It is important that we expand our understanding of side-channels and explore potential mechanisms that might be used to implement them.

In this paper we introduced three novel side-channel communication mechanisms: speed modulation, timing modulation and pin hijacking. We used these approaches to highlight an aspect of side-channels that, to the best of our knowledge, has been previously unexplored: the potential of an FPGA-based side-channel receiver. A side-channel receiver is significant because, taken together with prior work on how FPGAs can emit side-channel information, it enables two-way communication.

Our experiments on prototype side-channel receivers demonstrated that we could achieve reliable and, potentially, high bandwidth communication with minimal overhead. While there are still many issues we would like to address, these results prove that reliable bi-directional side-channel communication is possible. This leads the way to practical white-hat applications such as post-fabrication bug fixes and IP watermarking. However, it also enables powerful black-hat attacks such as conditionally triggered covert channels.

9. References

[1] Adamov, A., Saprykin, A., Melnik, D. and Lukashenko, O. The problem of Hardware Trojans detection in System-on-Chip. *International Conference of CAD Systems in Microelectronics* 2009. 178 – 179.

[2] Agrawal, D., Archambeault, B., Rao, J. R. and Rohatgi, P. The EM Side–Channel(s):Attacks and Assessment Methodologies. *International Workshop on Cryptographic Hardware and Embedded Systems* 2002.

[3] Agrawal, D., Baktır, S., Karakoyunlu, D., Rohatgi, P. and Sunar, B. Trojan Detection using IC Fingerprinting. *IEEE Symposium on Security and Privacy*. 296 – 310.

[4] Bar-El, H. "*Introduction to Side-channel Attacks*," White Paper, http://www.discretix.com

[5] Bar-El, H., Choukri, H., Naccache, D., Tunstall, M. and Whelan, C. The Sorcerer's Apprentice Guide to Fault Attacks. Proceeding of IEEE, Vol. 94, No. 2, (Feb. 2006), 370 – 382.

[6] Becker, G. T., Kasper, M. and Paar, C. 2010. Side-Channel based Watermarks for IP Protection. *International Workshop on Constructive Side-Channel Analysis and Secure Design* 2010.

[7] Bittner, R. The Speedy DDR2 Controller For FPGAs. *International Conference on Engineering of Reconfigurable Systems & Algorithms*. 2009.

[8] Chari, S., Rao, J. R. and Rohatgi, P. 2002 Template Attacks. *International Workshop on Cryptographic Hardware and Embedded Systems* 2002

[9] Collins, D. DARPA "TRUST in IC's" Effort. *Microsystems Technology Symposium Enabling the Future*. 2007.

[10] De Mulder, E., Buysschaert, P., Ors, S.B., Delmotte, P., Preneel, B., Vandenbosch, G. and Verbauwhede, I. Electromagnetic Analysis Attack on an FPGA Implementation of an Elliptic Curve Cryptosystem. *International Conference on Computer as a Tool*. EUROCON 2005. 1879 – 1882.

[11] *The I2C-bus Specification Version 2.1*. NXP, Jan. 2000.

[12] Lenz, M., Striedl, G. and Fröhler, U. 2000. *Thermal Resistance: Theory and Practice*. Infineon Technologies AG. (Jan. 2000)

[13] Lin, L., Kasper, M., Güneysu, T., Paar, C. and Burleson, W. Trojan Side-Channels: Lightweight Hardware Trojans through Side-Channel Engineering. *International Workshop on Cryptographic Hardware and Embedded Systems* 2009. 382 – 395.

[14] Lopez-Buedo, S., Garrido, J. and Boemo, E. I. Dynamically Inserting, Operating, and Eliminating Thermal Sensors of FPGA-Based Systems. *IEEE Transactions on Components and Packaging Technologies*, Vol. 25, No. 4, (Dec. 2002), 561 – 566.

[15] *Virtex-5 FPGA Packaging and Pinout Specification UG195 (v4.7)*. Xilinx Inc., Dec. 2009.

[16] *Part Number CP85338*, datasheet, V-Infinity Inc., Aug. 2009

[17] Zick, K. M. and Hayes, J. P. On-line sensing for healthier FPGA systems. *International Symposium on Field Programmable Gate Arrays.* 2010. 239 – 248.

[18] Ziener, F., Baueregger, F. and Teich, J. Using the Power Side Channel of FPGAs for Communication. *International Symposium on Field Programmable Custom Computing Machines.* 2010. 237-244.