Using Dynamic Reconfiguration to Implement High-Resolution Programmable Delays on an FPGA

Etienne Bergeron
Marc Feeley
DIRO, Université de Montréal
{*bergeret,feeley*}*@iro.umontreal.ca*

Marc-Andre Daigneault
Jean Pierre David
GRM, École Polytechnique de Montréal
{*marc-andre.daigneault,jpdavid*}*@polymtl.ca*

***Abstract*—A digital circuit can be viewed as a network of transistors switching between low and high voltages. These transistors and the wires interconnecting them cause delays in signal propagation. In most cases, designers aim to minimize the delays in order to increase processing speed. Nevertheless, some applications such as delay lines, time to digital converters, asynchronous logic and others require the ability to precisely control a delay between two points in a circuit. This paper proposes a novel way to control the delays in an FPGA by dynamically configuring the routing matrices to build a path with the required delay and to calibrate the delays. Such low-level configuration is possible with a dynamic reconfiguration library we developed for Xilinx FPGAs. Our experiments on Virtex-II Pro devices show that any differential delay in a range of 947ps can be reached with a precision of +/- 18ps.**

I. Introduction

Field-Programmable Gate Array (FPGA) technology has matured a long way since its beginnings in the early 80's. Time-to-market is an increasingly important factor and, due to their high density, which is equivalent to millions of logical gates, FPGAs are gaining importance in many applications. Furthermore, as some devices allow for run-time reconfiguration, new concepts such as virtual hardware have emerged and much research has been conducted in the area of dynamically configurable hardware. These concepts could be the premises of the next computing revolution since manufacturers such as Intel, AMD and HP seem to consider using them or interfacing to them. However, modern work in this area faces a critical problem: the lack of tools.

Over the years, several research projects in the area of dynamic reconfiguration have used the JBits SDK [GLS99] for the Virtex FPGAs manufactured by Xilinx. One idea behind using a library such as JBits is to get a finer granularity level in the design process and to reduce the limitations imposed by Xilinx's standard design flow. But as the technology kept progressing with the Virtex-II Pro, Virtex-4 and Virtex-5 families (all offering the capability of run-time reconfiguration), no complete tool or library such as JBits has been made available to designers and researchers to keep up with theses advances.

In a previous work, we have been able to understand the mapping between an FPGA bitstream and the programming points in Xilinx FPGAs [BFD07]. Since then, other researchers have published similar information [NR08]. Xilinx is also keenly interested in developing tools in this field [LBM+06]. Undoubtedly, low-level dynamic configuration will take an important place in the near future. In this paper, we present a new advance towards bridging the gap between the low-level dynamical configuration possibilities offered by modern Virtex FPGAs and current development tools. We developed a compact C library and related tool set for low-level dynamic reconfiguration, which currently supports Virtex-II Pro and Spartan-3 devices. In order to illustrate some of the new possibilities offered by this library, we present an application to delay management in a Virtex-II Pro FPGA.

Some applications require precise and known delays. This issue has already been addressed for clock signals by using programmable phase lock loops which change the phase of a clock signal. But this approach relies on the cyclic nature of clocks. Applications such as Time to Digital Converters (TDC), delay lines and asynchronous logic require the ability to measure and possibly modify the delay between two points in a circuit. These applications have always been confronted to a range/resolution dilemma and seldom FPGA implementations are mentioned in the literature [Kho06]. Indeed, even if current FPGAs offer many degrees of freedom to adjust the delays, proprietary tools are designed to optimize them, i.e. to minimize the delay of selected (critical) paths.

From a more general point of view, some applications require to control the configuration of an FPGA at the finest granularity, which is currently impossible with the standard design flow. The tools and methodology we developed offer such a fine control and the ability to modify the design's configuration at run-time. The experiments conducted on a Virtex-II Pro FPGA demonstrate that with dynamic reconfiguration it is possible and easy to have a very fine control on the delays by taking advantage of the architecture of the switch matrices. Our approach, which allows very precise programmable delay circuits, could lead to full FPGA integration replacing external dedicated devices.

Section 2 presents the library, which is part of our dynamic reconfiguration framework. Sections 3 and 4 explain the presence of delays in FPGAs and propose a way to control them. Section 5 gives the implementation details concerning delay measurement and calibration, and gives experimental results.

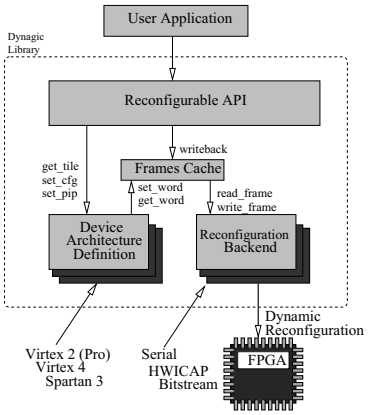

Figure 1. Dynamic reconfiguration framework

II. Dynamic Reconfiguration Framework

The DYNAGIC group [BDFD] is implementing development tools for dynamic applications. A tool of central importance is the `dynagic` library which abstracts the reconfiguration mechanism of Xilinx FPGAs. The purpose of this library is to allow an alternative design flow to the one offered by Xilinx [Xil04]. The main limitation of the Xilinx's flow is that it does not allow dynamic generation of modules; all modules must be statically generated.

Our library supports on-the-fly construction of fine-grained modules at the same level of abstraction as FPGA Editor. The developer can activate/deactivate each programmable point by using a C API.

As the design in Figure 1 shows, the library is divided so that it is possible to plug specific modules that abstract parts of the required functionality.

The *device architecture definition* provides functionalities related to a specific FPGA family (e.g. Virtex-2, Virtex-2 Pro, Spartan-3, Virtex-4...). The *reconfiguration backend* provides raw access to the FPGA configuration memory. As dynamic applications may run in different contexts (embedded or on a host), the library has different implementations. The *frame cache* provides faster reconfiguration performance by caching read/write operations. Thus, by using the reconfiguration API, it is possible to construct a dynamic application with its specific needs.

Figure 2 shows an example of using the library. The `dg_open` function initializes the library. The *reconfiguration backend* (ICAP, JTAG, serial port, ...) is dynamically selected depending on the first program argument (`argv[1]`). Thus it allows the same program to run on a host PC or on the embedded processor. The `dg_get_tile` function returns a handle of the tile at a specific location of the device grid. The `dg_get_site` function returns a handle to a site in a specific tile. The `dg_set_cfg` sets the value of a programmable point at a specific site. The `dg_set_pip` function activates a programmable interconnect point (PIP) in the switch matrix of a specific tile.

```
#include "dynagic.h"

#define LUT_NOT_G2 ~0x3333
#define LUT_F3     ~0xF0F0

int main(int argc, char** argv) {
  dg_system_t sys;
  dg_tile_t tile;
  dg_site_t slice;

  dg_open(&sys,argv[1]);
  dg_capture(&sys);

  tile  = dg_get_tile(&sys, 1, 1);
  slice = dg_get_site(&sys, tile, V2_COMP_SLICE1);

  dg_arch_set_lut(slice, V2_RESS_G,     LUT_NOT_G2);
  dg_arch_set_cfg(slice, V2_RESS_GYMUX, V2_VAL_G);
  dg_arch_set_cfg(slice, V2_RESS_YUSED, V2_VAL_0);
  dg_arch_set_lut(slice, V2_RESS_F,     LUT_F3);
  dg_arch_set_cfg(slice, V2_RESS_FXMUX, V2_VAL_F);
  dg_arch_set_cfg(slice, V2_RESS_XUSED, V2_VAL_0);

  dg_arch_set_pip(tile, V2_WIRE_Y1, V2_WIRE_DY1);

  dg_frames_cache_writeback(&sys);
  dg_close(&sys);
}
```

Figure 2. `dynagic` library example

III. Design considerations for the delay circuit

A digital signal is transmitted from a source component to a target component by routing it through a chain of intermediate components. Each one propagates the signal to the next component in the chain until the target is reached. The FPGA's configuration determines which route the signal will take. Each component on a route contributes to the total delay from source to target. The delay depends on wire length and capacitance, transistor switching speed, temperature, and other factors.

On a Xilinx Virtex FPGA, the intermediate components are either wires, logic elements, or switch matrices (which probably contain wires, pass transistors and buffers). A common approach of implementing a delay on Virtex devices is to route the signal through a chain of N look-up tables (LUTs). The LUTs have a propagation delay of roughly 250ps, so the total delay is roughly $N \times 250$ps plus the delay of the routes needed to interconnect the LUTs in a chain and connect them to the source and destination. Each connection typically adds a delay in the hundreds of picoseconds, but sometimes more than a nanosecond. Some tools, FPGA Editor in particular, can give a maximum delay for these connections but not the actual or expected delay. So LUT-based delays suffer from a rather coarse resolution, and a substantial uncertainty on the actual delay.

The approach we developed uses only the switch matrix to implement the delay. The switch matrix is a component which can be configured to transmit a signal from one of several input pins to one (or more) of its output pins. The pins of the logic elements and the inter-tile network wires are connected to these inputs and outputs. For a given pair of input and output pins there is typically a very large number of ways to route the

signal within the switch matrix. This is due to the architecture of the switch matrix which can route a signal from an input pin to an output pin unconnected externally, and *bounce* the signal to another output pin (see Figure 3).

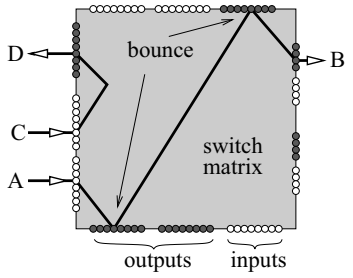

Figure 3. A switch matrix configured with the routes $A \rightarrow B$ and $C \rightarrow D$ of depth 3 and 1 respectively

We define the *depth* of a route as the number of switch matrix wire segments that are used. The depth is also the number of PIPs that must be activated in the configuration to create this route. The total delay of a route depends on the choice of wire segments and is roughly proportional to its depth. There is a considerable disparity in the delay contributed by the switch matrix wire segments. Given the high number of possible routes, many different delays can be obtained. A table of routes up to a certain depth and the associated delay could be created and used by a tool to achieve, between an input and output pin, a delay as close as possible to the desired delay in a certain time range.

The actual delay of a route is influenced by device properties which can vary from part to part, and by environmental parameters such as temperature which can vary over time. To achieve repeatability and accuracy it is desirable to calibrate the device during its operation. To reduce the drift over time caused by temperature changes the delay for the set of routes in the table must be measured repeatedly throughout the device's operation. To do this we use dynamic reconfiguration.

IV. Switch matrix routes

The switch matrix routes used by our method are intentionally non-optimal, so they cannot be generated with standard tools. The set of pins and PIPs contained in a switch matrix are described in the device's XDL report generated with the command "`xdl -report -pips ...`". We have used that information and a simple depth first search to create a database of all the possible routes from inputs to outputs up to a depth of 9.

To test our approach we have generated the routes between the output of LUT G in SLICE1 of a CLB (pin Y), and the inputs F1, F2, F3 and F4 of LUT F in the same slice. There are 546 different routes with a depth ≤ 9. Other pins could have been used, but these were chosen because they are easy to connect to other components. The signal to delay must be routed to one of the inputs of LUT G and the delayed signal will be on the output of LUT F (pin X). The switch matrix must be configured to implement the appropriate route for the desired delay, LUT F must be configured to propagate the output of the switch matrix to its output pin X, and LUT G must be configured to propagate the appropriate input to the output pin Y. This approach allows the total delay to be adjusted without changing the routing of the signal to LUT G and from LUT F (see Figure 4). Note that the routing to LUT G and from LUT F can be avoided in some cases by using these LUTs to perform useful operations. Avoiding the LUTs altogether is also possible, but it increases the difficulty of delay measurement and calibration.

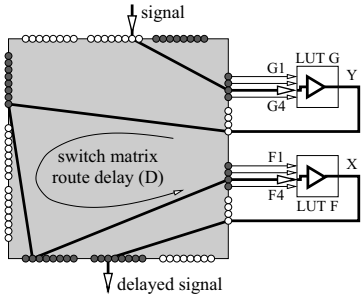

Figure 4. Configuration of the switch matrix and LUTs when a signal is being delayed

V. Delay measurement and calibration

The switch matrix delay for a given route can be measured with a ring oscillator and a ripple binary counter driven by the oscillator (see Figure 5). A ring oscillator could be built by configuring LUT G as an inverter and LUT F as a buffer, and sending the output from pin X to an input of LUT G. The signal at pin X is also sent to the ripple counter. This oscillator might exceed the maximal switching frequency of the FPGA so to be safe we added 2 other buffers in the feedback loop to slow down the oscillator to under 400MHz. Given an accurate time base, such as an external crystal, it is a simple matter to read the counter after a given route has been configured, and to read it again after T seconds. The difference X is related to the period P of the oscillator by the formula: $P = T/X$. The half period of the oscillator is equal to the switch matrix routing delay D plus R, the propagation delay of the LUTs and the routing to connect them. So $D = P/2 - R$. It must be observed that, while R is an unknown, it is unaffected by the switch matrix routing. This means that the delay difference between two routes can be obtained. This is often the most important information (an absolute time is rather useless when the routing delay of the incoming signal can't be known precisely). If needed R can be approximated by choosing a route with a delay that is known to be short (using the maximal delay reported by FPGA Editor). If we use the shortest route (Y1 $\rightarrow$ F3_B1 $\rightarrow$ F3_B_PINWIRE1), which has a reported maximal delay of 53ps, the half period measured is 1364ps, so R = 1338ps +/- 27ps.

During calibration, the tile containing the delay line and a few other CLBs are reconfigured dynamically to implement

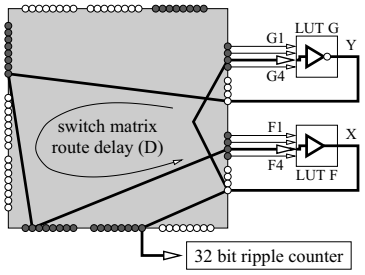

Figure 5. The configuration of the switch matrix and LUTs when the route's delay is being measured

the ring oscillator and ripple counter. The value of the counter is accessed by the processor through the ICAP. It takes less than 10 seconds to measure the half period of over 500 routes. This could be made faster using a dynamically reconfigured dedicated control circuit. To verify the stability of the delays we performed the measurements 20 times. The measured half period for a given route varied by at most 1ps. We also performed a linear regression to try to explain the half period as a linear function of the maximal delay (M) reported by FPGA Editor. The regression gives $P/2 = 1317ps + 0.598 \times M$. Figure 6, which plots the residual errors, shows that there is a good correlation between M and D.

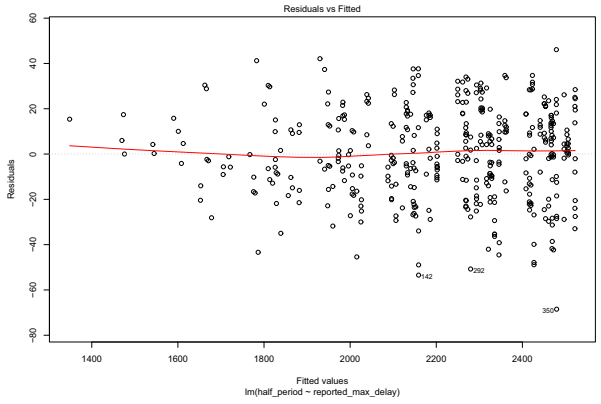

Figure 6. Residual errors for $P/2 = 1317ps + 0.598 \times M$

Table I gives the switch matrix route delay (D) obtained by substracting 1338ps (R) from the half period measured for the routes. To save space, out of the 546 routes measured, only the 10 shortest delay routes and the 2 longest are shown. Switch matrix route delays between 266ps and 1213ps, a range of 947ps, are spaced at most 36ps apart. This means that for any target delay in this interval a route exists whose delay is no more than 18ps away from the target. The precision increases substantially if we restrict the delay to a subinterval in the larger delays. For instance, between 708ps and 1213ps, a range of 505ps, the error is +/- 8ps. This higher precision is due to the increasing density of route delays for longer routes.

26ps	Y1 F3_B1 F3_B_PINWIRE1
138ps	Y1 W2BEG4 F3_B1 F3_B_PINWIRE1
138ps	Y1 S2BEG4 F3_B1 F3_B_PINWIRE1
153ps	Y1 OMUX6 F3_B1 F3_B_PINWIRE1
207ps	Y1 S2BEG2 F1_B1 F1_B_PINWIRE1
208ps	Y1 E2BEG1 F1_B1 F1_B_PINWIRE1
266ps	Y1 OMUX6 W2BEG2 F3_B1 F3_B_PINWIRE1
268ps	Y1 OMUX4 W2BEG2 F3_B1 F3_B_PINWIRE1
273ps	Y1 OMUX15 E2BEG9 F4_B1 F4_B_PINWIRE1
279ps	Y1 OMUX15 N2BEG9 F4_B1 F4_B_PINWIRE1
...	
1209ps	Y1 E2BEG1 BX0 BY3 BX3 BY2 BX1 BY1 F1_B1 F1_B_PINWIRE1
1213ps	Y1 S2BEG2 BX0 BY3 BX3 BY2 BX1 BY1 F1_B1 F1_B_PINWIRE1

Table I
MEASURED SWITCH MATRIX ROUTE DELAYS FOR ROUTES BETWEEN Y AND F1, F2, F3 AND F4.

The number of routes increases exponentially with the route depth, whereas the route delay increases roughly linearly with the route depth.

VI. CONCLUSION

Switch matrices in Virtex FPGAs allow numerous ways to route a signal for a given pair of input and output points. Standard tools automatically configure these matrices to attempt to get the minimal delay for a given set of routing constraints. We have presented a dynamic reconfiguration library with a flexible and efficient API allowing the dynamic configuration of an FPGA at the finest level of granularity. Thanks to this library, we have been able to generate custom routes in the switch matrices and measure their delays. This information was used to implement a way of controlling the delays in an FPGA with a precision varying from 8ps to 18ps, depending on the required delay range. This work gives a taste of the previously unimplementable applications now achievable with our library.

REFERENCES

[BDFD] Etienne Bergeron, Marc Andre Daigneault, Marc Feeley, and Jean Pierre David. Dynagic Web Page. http://www.dynagic.org.

[BFD07] Etienne Bergeron, Marc Feeley, and Jean Pierre David. Toward On-Chip JIT Synthesis on Xilinx Virtex-II Pro FPGAs. In *50th International Midwest Symposium on Circuits and Systems/5th International Northeast Workshop on Circuits (MWCAS/NEWCAS), Montreal, Canada*, August 2007.

[GLS99] S. Guccione, D. Levi, and P. Sundararajan. JBits: A Java-based interface for reconfigurable computing, 1999.

[Kho06] Amir Mohammad Amiri; Mounir Boukadoum; Abdelhakim Khouas. Low Dead Time, Multi-hit FPGA-Based Time-to-Digital Converter. *Circuits and Systems, 2006 IEEE North-East Workshop on*, pages 29–32, June 2006.

[LBM+06] Patrick Lysaght, Brandon Blodget, Jeff Mason, Jay Young, and Brendan Bridgford. Invited Paper: Enhanced Architectures, Design Methodologies and CAD Tools for Dynamic Reconfiguration of Xilinx FPGAs. In *FPL*, pages 1–6, 2006.

[NR08] Jean-Baptiste Note and Eric Rannaud. From the Bitstream to the Netlist. In *Sixteenth ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2008.

[Xil04] Xilinx. Two Flows for Partial Reconfiguration: Module Based or Difference Based. Technical report, Xilinx, September 2004.