# Memory-efficient and fast run-time reconfiguration of regularly structured designs

Brahim Al Farisi, Karel Heyse, Karel Bruneel and Dirk Stroobandt
*Ghent University, ELIS Department*
*Sint-Pietersnieuwstraat 41, 9000 Gent, Belgium*
{*Brahim.AlFarisi, Karel.Heyse, Karel.Bruneel, Dirk.Stroobandt*}@*UGent.be*

***Abstract*—Previous work has shown that run-time reconfiguration of FPGAs benefits greatly from the use of Tunable LUT (TLUT) circuits. These can be rapidly transformed into a specialized LUT circuit and are also very memory efficient when representing regularly structured designs, where the same hardware module is instantiated many times. However, the memory requirements and reconfiguration time of a run-time reconfigurable application are also dependent on the reconfiguration mechanism. In this paper, we will show that the memory requirements of conventional ICAP reconfiguration grow very fast with the number of modules, resulting in excessive memory usage. We propose to use Shift-Register-LUT (SRL) reconfiguration which is faster and results in a memory usage that is independent of the number of modules.**

***Keywords*-FPGA; Run-time Reconfiguration; Tunable LUT circuit; ICAP; SRL;**

## I. Introduction

The inherent reconfigurability of SRAM-based FPGAs enables the use of configurations optimized for the problem at hand. Optimized configurations are smaller and faster than their generic counterparts and therefore use the FPGA's resources more efficiently. However, at some point the problem at hand will change and valuable system resources will be needed to generate a new configuration and reconfigure the system's FPGA. These two run-time reconfiguration (RTR) tasks are performed by a subsystem we call the *configuration manager* (CM), usually a CPU. Using RTR, the designer trades FPGA resources for CM resources.

RTR can be beneficially used for *data folding* applications [1], where in each reconfiguration interval the design is optimized for a specific set of input values, called *parameters*. We have called this technique *dynamic data folding*. The TMAP tool flow, presented in [2], automatically maps dynamic data folding applications to a self-reconfiguring platform. It starts from a parameterizable HDL description that describes the functionality of the reconfigurable module. It automatically generates a LUT circuit with fixed routing where the truth table bits of some of the look-up tables (LUTs) are expressed as Boolean functions of the parameters. This circuit is called a Tunable LUT (TLUT) circuit and can be rapidly transformed into a specialized LUT circuit by evaluating the Boolean functions for a specific set of parameter values. The fixed LUT structure is implemented in the FPGA fabric and the Boolean functions are compiled to a specialization procedure that is tailored to the CM. Every time the parameters change, this procedure generates new truth tables for the LUTs, and reconfigures them.

In regularly structured designs the same hardware module is instantiated many times. The function of every module is the same, only the inputs that have to be processed are different. These designs are a very important class of FPGA applications, that rely heavily on massively parallel computation. An example of a regularly structured design is a FIR filter, where the instantiated modules are all multipliers with different multiplication coefficients. Using a TLUT circuit as a module provides a means for memory-efficient and fast run-time reconfiguration of regularly structured designs. The fixed LUT structure is instantiated many times on the FPGA, while the Boolean functions only have to be stored once in the CM and evaluated separately for every parameter instance.

However, the memory requirements and reconfiguration time of a run-time reconfigurable application not only depend on the calculation of the configuration bits but also on the reconfiguration mechanism. The convential run-time reconfiguration mechanism uses the Internal Configuration Access Port (ICAP) as an interface to the configuration memory of the FPGA. Because of the randomness of the placement process, the LUTs of the different modules are scattered across the FPGA. Thus, in the configuration memory the regularity of the design is lost, which results in excessive memory usage in the CM.

When only LUTs need to be reconfigured, it has been previously proposed to use Shift-Register-LUTs (SRLs) as a fast run-time reconfiguration mechanism [3] [4] [5] [6]. In an SRL, the truth table configuration bits of the LUT are also arranged as a shift register of which the input and the output are accessible from the configurable routing.

In this paper we make following novel contributions:

1) We point out that using a TLUT circuit as a module of regularly structured designs provides a means for memory efficient and fast run-time reconfiguration.
2) We show that SRL reconfiguration can be used to retain the regularity of the design in the reconfiguration interface, making the memory usage in the CM independent of the number of modules.
3) We quantify the speedup of SRL reconfiguration over ICAP for an adaptive filtering application.
4) SRL reconfiguration introduces extra nets in the design

that could influence the timing of the design. We examine this influence when increasing the number of modules of an adaptive filtering application.

The paper is organized as follows. A quick review of the TMAP dynamic data folding toolflow is presented in section II. In section III, the different mechanisms for reconfiguring LUTs, namely SRLs and ICAP, are described. The memory usage and reconfiguration time of ICAP and SRL reconfiguration for regularly structured designs is described in section IV. Finally, in section V we will show that for an adaptive filtering application the SRLs have better memory efficiency and reconfiguration time than the ICAP, while the impact of introducing the SRLs is limited.

## II. Dynamic Data Folding

Dynamic data folding (DDF) applications have two types of inputs that are treated differently: fast changing inputs (*normal* inputs) and slow changing inputs (*parameter* inputs). Instead of building generic circuitry where both types of inputs are regular input signals, we build a dynamic data folding system where only the normal inputs are inputs to a reconfigurable module implemented in the FPGA fabric. The parameters are inputs to a second subsystem, the configuration manager (CM), in our case an instruction set processor (ISP). Every time the parameters change, the CM specializes the reconfigurable module for the new parameter values. Once specialized, the module is ready to process the fast changing input data. The reason to build a DDF system is that the reconfigurable module can be implemented more efficiently in the FPGA fabric than the generic circuitry.

With convential FPGA tools only handcrafted DDF systems are possible [7] [6]. The TMAP tool flow on the other hand automatically maps dynamic data folding applications to a self-reconfiguring system [2]. The input of the tool chain is a behavioral description of the functionality in which a distinction is made between normal inputs and the parameter inputs. The output is a Tunable LUT (TLUT) circuit that consists of a fixed LUT-structure and a Boolean circuit we call the Partial Parameterizable Configuration (PPC). The PPC describes the Boolean dependency of the truth table bits on the parameters as a Boolean circuit that consists of AND and inverter gates. This is also called an AND-Inverter Graph (AIG) [8]. As an example we chose the selection bits of a 4-input multiplexer as parameters and mapped it to 3-LUTs. The resulting fixed LUT structure and AIG of the PPC are shown in Figure 1. We note that making a generic 4-input multiplexer with 3-LUTs takes 6 LUTs, while this datafolded version only takes 2 LUTs.

The fixed LUT circuit can be placed and routed on the FPGA fabric using conventional tools. The PPC is compiled to an evaluation function that has to be carried out by the CM. More specifically, the evaluation function consists of C-code that can run on an instruction set processor (ISP). From the locations of the LUTs on the FPGA and the evaluation

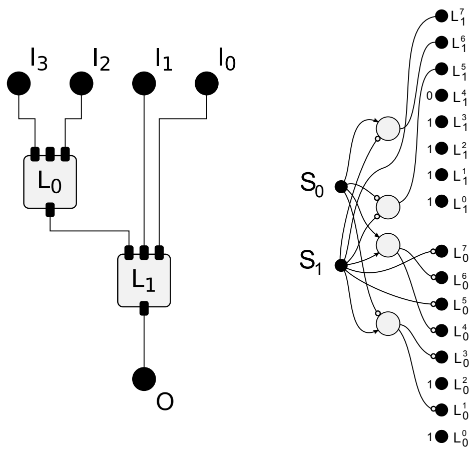

Figure 1: The fixed LUT structure and AIG of the PPC for the 4-input multiplexer example.

function of the PPC the *specialization* procedure is synthesized. The specialization procedure takes the parameters as arguments, generates new truth tables for the reconfigurable module and writes them in the configuration memory. The specialization procedure thus consists of an *evaluation* of the PPC and a *reconfiguration* of the truth tables of the fixed LUT circuit.

## III. LUT reconfiguration

### A. ICAP reconfiguration

The conventional run-time reconfiguration mechanism uses the Internal Configuration Access Port (ICAP). This is an interface to the entire configuration memory and thus also to the truth table bits of the LUTs. Xilinx provides a software interface for the ICAP. This consists of the HWICAP peripheral that can be attached to a processor's bus and the accompanying driver program [9].

The FPGA configuration memory of recent Xilinx devices is arranged as frames that are tiled on the device. A frame is the smallest addressable segment of the configuration memory. When using the ICAP, all operations must therefore act upon complete configuration frames [10].

### B. SRL reconfiguration

When only LUTs have to be reconfigured, as is the case in a TLUT circuit, it is also possible to use shift-register-LUT (SRL) reconfiguration. In an SRL the configuration bits of the truth table are arranged as a shift register of which the input and the output are accessible from the configurable routing. Therefore the truth table configuration bits can be changed by shifting in a new truth table. This idea is not novel and has been proposed various times in previous literature [3] [4] [5] [6].

In order to make the truth table bits of multiple TLUTs accessible from the CM, we group them and arrange each

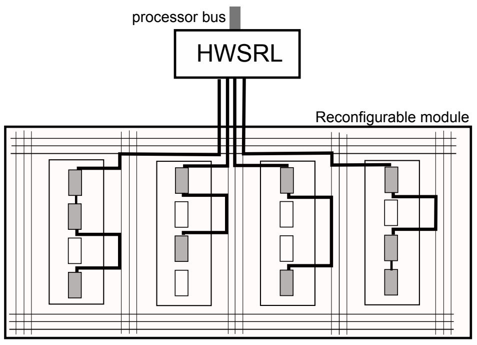

Figure 2: SRL reconfiguration with 4 reconfiguration paths.

group as a larger shift register, called a *reconfiguration path*, by connecting the shift out of a TLUT to the shift in of the next TLUT. The shift in of the first TLUT of each reconfiguration path is connected to the HWSRL, which replaces the HWICAP and also interfaces to a processor's bus. The HWSRL is basically a FIFO that buffers the reconfiguration data, with logic added to start and stop the reconfiguration. On the side of the processor's bus the FIFO is 32 bit wide, while on the side of the reconfiguration paths the width depends on the number of reconfiguration paths. An example of reconfiguration with 4 reconfiguration paths using the HWSRL is shown in Figure 2. We have three degrees of freedom when constructing the SRL reconfiguration infrastructure. We can vary (i) the number of reconfiguration paths, (ii) how TLUTs are partitioned into reconfiguration paths, and (iii) the order of the TLUTs in the reconfiguration paths. These degrees of freedom can be used to to increase the reconfiguration speed and minimize the impact of SRL reconfiguration on the maximum clock frequency of the design. One of the novelties of this paper is that one can use the 2 last degrees of freedom to minimize the memory usage when reconfiguring regularly structured designs. This is explained further in section IV-B.

## IV. Memory efficiency and reconfiguration time

As described in section II, the TMAP toolflow automatically maps a dynamic data folding application onto a TLUT circuit. TLUT circuits are very memory efficient, since the specialized LUT circuits are not stored separately but as Boolean functions of a parameter. We also note that the routing and LUT truth table configuration bits of recent Xilinx Virtex FPGAs reside in different configuration frames [10]. Since the routing of a TLUT circuit is fixed,

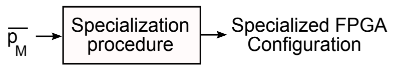

Figure 3: The specialization procedure in the case of regularly structured designs.

```
function specializationICAP ( parameterList )
  for frameAddress in frameAddressArray:
    Frame = getConstantData(frameAddress);
    for each Tlut in Frame:
      ( frameIndex, tlutPPCFunction, parameterIndex ) =
      getLUTInfo(Tlut);
      Frame [frameIndex] =
      evaluateTLUT(tlutPPCFunction, parameterIndex);
    configureFPGA(Frame, frameAddress);
```

Figure 4: Pseudo code for ICAP specialization.

no information has to be stored in the CM concerning the routing.

In regularly structured designs the same hardware module is instantiated many times. The function of every module is the same, only the inputs that have to be processed are different. Using a TLUT circuit as a module of regularly structured designs provides a means for memory efficient and fast run-time reconfiguration of such designs. Because the function of all the modules is the same, the PPC only has to be stored once in the specialization procedure.

The specialization procedure in the case of regularly structured designs is shown in Figure 3. During run-time this procedure takes in a list $\overline{p_M}$ containing the parameter values of the different modules and generates a specialized FPGA configuration. Because the PPC only has to be stored once, the memory requirements for the evaluation of the PPC is constant and independent of the number of modules. However, the memory requirements of the specialization procedure are also dependent on the information needed for reconfiguration. In this section we will compare the memory usage of the specialization procedure when using ICAP and SRL reconfiguration. We will show that for ICAP reconfiguration the memory usage is dependent on the number of modules, which results in excessive memory usage when the number of modules is increased. For SRL reconfiguration we will show that we can construct the reconfiguration paths in such a way that no information needs to be stored for reconfiguration. In this paper we are particularly interested in how the memory usage, expressed in bits, of the specialization procedure scales with the number of modules. In every section we will also briefly discuss the reconfiguration time. The evaluation time of the PPC is outside the scope of this paper.

### A. ICAP reconfiguration

As described in section III-A, the ICAP reconfiguration interface processes the reconfiguration data per frame. The pseudo code for the specialization procedure is in this case given in Figure 4. We note that the size of the C code that implements this relatively simple pseudo code will be small and independent of the number of modules. To estimate the memory usage when increasing the number of modules, we thus neglect the size of the actual specialization procedure and concentrate only on the amount of information we need in the specialization procedure.

The placement process distributes the TLUTs of different modules irregularly across various different configuration frames. This is illustrated in Figure 5(a) for a regularly structured design with 2 modules and 3 TLUTs per module that are scattered across 2 frames. This scattering has several consequences. First of all, information has to be stored containing the locations of the TLUTs in the configuration memory. This location information consists of a frame address and an index in the frame. We denote the number of modules $M$, the number of TLUTs per module $L_M$, the number of frames F, the number of LUTs per frame $L_F$ and the number of bits to store the frame address $b_A$. The number of bits needed to store this information is then $F \cdot b_A + M \cdot L_M \lceil log2(L_F) \rceil$.

Second, the TLUTs of one module are also distributed irregularly across several different frames. To accomodate evaluation on a frame basis, as is shown in Figure 4, the PPC of the module must be adjusted to accomodate evaluation per TLUT. Per TLUT of the module we thus create a separate PPC. Per TLUT one also has to store the function to be called to evaluate the PPC of the TLUT and a pointer to the parameter value of the respective module in the parameter list $\overline{p_M}$. The main consequence is that Boolean gates of the PPC of the module, that are reused across TLUTs will have to be duplicated. This increases the memory requirements for the compiled C-code that evaluates the PPC. Since the function of all the modules is the same this C-code also only has to be stored once. For example, all the TLUTs with name 'TLUT1' in Figure 5 will use the same PPC. We denote $PPC_{LUT}$ the sum of the size of the compiled C-code of the PPCs of the different TLUTs. The number of bits needed to store this information is then $PPC_{LUT} + M \cdot L_M \cdot (\lceil log2(M) \rceil + \lceil log2(L_M) \rceil)$.

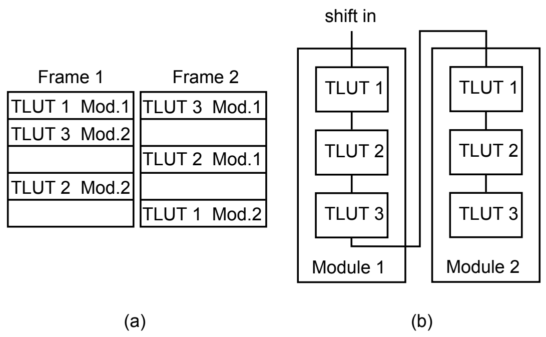

Figure 5: The LUTS of a regularly structured design irregularly scattered in the ICAP configuration memory (a) and regularly placed in an SRL reconfiguration path (b).

Finally, when reconfiguring the TLUTs, the constant data in the frames is also rewritten. One option is to keep this constant data in the memory of the CM. This is very memory consuming, but has the fastest reconfiguration time. Since the configuration memory can also be read we can also apply a read-modify-write strategy, where reconfiguration time is sacrificed for more memory efficiency.

$$M_W = PPC_{LUT} + F \cdot (b_A + b_F) + M \cdot L_M \cdot b_L \quad (1)$$

$$M_{RMW} = PPC_{LUT} + F \cdot b_A + M \cdot L_M \cdot b_L \quad (2)$$

where $b_L$ is given by $\lceil (log2(L_F)) \rceil + \lceil (log2(L_M)) \rceil + \lceil (log2(M)) \rceil$ and $b_F$ is the number of bits per frame. In both equations (1) and (2) we see that the memory usage grows with the number of modules.

The reconfiguration time is given for both versions of ICAP reconfiguration, in equations (3) and (4).

$$T_W = \frac{F \cdot b_F \cdot T_{ICAP}}{D} \quad (3)$$

$$T_{RMW} = 2 \cdot T_W \quad (4)$$

As mentioned above $F$ and $b_F$ are the number of frames and the number of bits in a frame. D is the width in bits and $T_{ICAP}$ is the clock period of the ICAP interface. The reconfiguration time of the read-modify-write version of ICAP reconfiguration is simply double that of the write-only version. The data has to be processed once when read and once when written, while the bandwidth of the ICAP when reading is the same as when writing [9].

### B. SRL reconfiguration

The scattering of the TLUTs (and the growth of memory usage with the number of modules) can be avoided using SRLs, as shown in Figure 5 (b). The degrees of freedom when constructing reconfiguration paths, as described in section III-B, can be used to retain the regularity of the design in the reconfiguration paths. We therefore choose the reconfiguration paths so that the TLUTs of one module are always coherent and the order of the different TLUTs in a module is the same. Of course there are still different ways to connect the different TLUTs. In this paper the ordering of the modules and the TLUTS in the modules is chosen ad-hoc. Further optimizations are possible.

By choosing the reconfiguration paths as mentioned above, one can take full advantage of the regularity of the design to minimize the memory usage of the specialization procedure. The pseudo code that represents the specialization procedure in this case is shown in Figure 6. It is important to understand that the order in which TLUTs in a module are specialized, is the same for all modules and corresponds with the order chosen in the SRL reconfiguration paths. That is why we can use the for loop in the pseudo code in Figure 6. As shown in equation (5), the only data that is stored is the evaluation function of the PPC. The main point

```
function SpecializationSRL ( parameterList )
  for parameter in parameterList :
    evaluatedModule = evaluateModule(parameter) ;
    configureFPGA(evaluatedModule);
```

Figure 6: Pseudo code for the SRL specialization procedure.

this paper makes is that combining TLUT circuits and SRL reconfiguration results in a very memory efficient method for run-time reconfiguration of regularly structured designs.

$$M_{SRL} = PPC_{Module} \quad (5)$$

Using SRLs also has advantages for the reconfiguration time. Only the LUTs that have to be reconfigured are put in the reconfiguration paths. Increasing the number of reconfiguration paths reduces the number of bits per reconfiguration path and increases the reconfiguration speed. In the SRL case, the reconfiguration speed can thus be tuned to the application requirements.

$$T_{SRL} = \frac{M \cdot L_M \cdot 2^K \cdot T_{shift}}{R} \quad (6)$$

The formula for obtaining the reconfiguration time in the case of SRL reconfiguration is given above in equation (6). The time needed to reconfigure the design is dependent upon the number of modules M, the number of TLUTs per module $L_M$, the number of inputs K of one SRL, the period $T_{shift}$ of the clock frequency at which the bits are shifted in and the number of reconfiguration paths R.

## V. Evaluation and experiments

We illustrate the ideas above by implementing a fully pipelined FIR filter, of which the coefficients are chosen as the parameters. The module in this case is an 8 by 8 bit multiplier of which one of the operands is a parameter. The parameter list $\overline{p_M}$, depicted in Figure 3, thus contains the specific coefficient values we want the FIR filter to be specialized for during run-time. It has been shown that such a run-time reconfigurable FIR filter is 40 % more area efficient than a generic FIR filter of which the coefficients are inputs to the FPGA [2]. We will compare the memory usage of the specialization procedure for ICAP and SRL reconfiguration while increasing the number of modules from 64 to 1024. Also the reconfiguration time and impact of SRL reconfiguration on the maximum clock frequency of the FIR filter are discussed.

The general characteristics of the FIR filter relevant for the equations from the previous section are $L_M = 24$, $PPC_{LUT} = 296kb$ and $PPC_{Module} = 136kb$. To obtain the values of $PPC_{LUT}$ en $PPC_{Module}$ we compiled the evaluation C-functions on a Microblaze v7.10.c [11]. The FIR characteristics that are dependent upon the number of modules are given given in Table I. We conduct this experiment on a Virtex4(xc4vlx100) [12] using ISE 10.1 software with default settings. The characteristics of the Virtex 4 ICAP relevant for the equations from the previous section are given in Table II.

Table I: Number of frames (F) and shift clock period ($T_{shift}$) for the different FIR filters.

| M | 64 | 128 | 256 | 512 | 1024 |
|---|---|---|---|---|---|
| F | 182 | 339 | 601 | 1095 | 1422 |
| $T_{shift}$ (ns) | 5,170 | 7,23 | 8,203 | 8,050 | 8,529 |
| max clk (MHz) | 198 | 190 | 186 | 156 | 92 |

Table II: Characteristics of a Virtex4 FPGA.

| K | $T_{ICAP}$ | D | $b_F$ | $L_F$ | $b_A$ |
|---|---|---|---|---|---|
| 4 | 10 ns | 32 bit | 1312 bit | 80 | 32 bit |

### A. Memory efficiency

In Figure 7 (a) the memory usage of ICAP reconfiguration, both the write-only and the read-modify-write version, and SRL reconfiguration is shown relative to the total memory of all BRAMs of the FPGA, which is 4320 Kb [12]. We see that the memory requirements of the write-only ICAP reconfiguration increase dramatically with the number of modules, consuming more than 60 % of the FPGAs BRAMs for the case with 1024 modules. Indeed, storing the constant data of the frames is very memory consuming. Even the more memory efficient read-modify-write ICAP version needs 20 % of the FPGA BRAMs. With SRL reconfiguration, on the other hand, even for a FIR filter with 1024 modules only 3% of the FPGAs memory is needed. As pointed out earlier, the memory usage for SRL reconfiguration is independent of the number of modules.

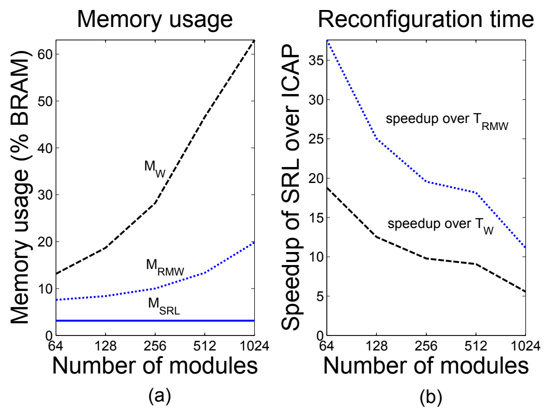

Figure 7: Influence of the number of modules on the memory usage and reconfiguration time.

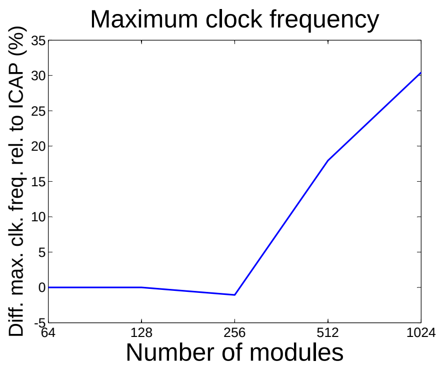

Figure 8: Influence of the SRL reconfiguration paths on the maximum clock frequency.

### B. Reconfiguration time

For the experiments conducted in this paper we assume a fixed number of 32 reconfiguration paths. Exploring different numbers of reconfiguration paths is outside the scope of this paper, and we thus choose the width of the SRL reconfiguration interface the same as the ICAP interface of the Virtex4. Of course, increasing the number of reconfiguration paths would further improve the results obtained for the reconfiguration time of SRL reconfiguration.

In Figure 7 (b) we can see that, in this case, at least a 6X and at most a 37X speedup over ICAP reconfiguration can be obtained when using SRL reconfiguration. The decrease in speedup as the number of modules increases has two reasons. The number of frames increases less than linearly with the number of modules, as opposed to the total number of TLUTs $M \cdot L_M$. A second reason is the occurence of routing congestion, that reduces the clock frequency at which the bits are shifted in. These two effects can be clearly seen in Table I. We point out that for ICAP reconfiguration a trade-off has to be made between speed and memory-efficiency. Using SRLs results in a reconfiguration process that is both memory-efficient and fast.

### C. Impact on maximum clock frequency

The maximum clock frequency of the designs without SRLs can be found in the last row of Table I. The decrease in maximum clock frequency as the number of modules increases is due to routing congestion. The FIR filter with 64 modules occupies 4 % of the slices of the FPGA, while the one with 1024 modules takes 87 %. In Figure 8 we can see the difference between the maximum clock frequency of the design with SRL reconfiguration paths and the design without, which is reconfigured using ICAP reconfiguration. We can clearly see that the negative impact of SRL reconfiguration on the maximum clock frequency is very limited. The decrease is never bigger than 1 %. In fact, as the number of modules increases, we would expect that due to increased routing congestion the maximum clock frequency to be worse when the SRL reconfiguration paths are present. The maximum clock frequency of the FIR filter however improves compared to ICAP reconfiguration. The presence of the SRL reconfiguration paths of course influences the packing and placement and routing process and in the case of the FIR filter this seems to be a very positive influence.

## VI. Conclusion and Future work

In this paper, we presented a memory-efficient and fast run-time reconfiguration method for regularly structured designs. This method combines the memory efficiency of TLUT circuits and SRL reconfiguration, resulting in a system where the memory usage is independent of the number of modules. For an adaptive filtering applicaton we also showed that SRL reconfiguration achieves at least a 6X speedup over ICAP reconfiguration. We also examined the impact of the SRL reconfiguration paths on the maximum clock frequency of this design for a Virtex4 FPGA. We showed that the decrease in maximum frequency never exceeds 1 % even in the case when the FPGA is almost fully occupied.

## References

[1] P. Foulk, "Data-folding in SRAM configurable FPGAs," in *Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines*, 5-7 1993, pp. 163 –171.

[2] K. Bruneel, F. Abouelella, and D. Stroobandt, "Automatically mapping applications to a self-reconfiguring platform," in *Proceedings of DATE*, 2009, pp. 964–969.

[3] I. O. Kennedy, "Implementation of low frequency finite state machines using the virtex srl16 primitive," in *FPL*, 2007, pp. 675–678.

[4] T. Sasao and H. Nakahara, "Implementations of reconfigurable logic arrays on FPGAs," in *FPT*, 2007, pp. 217–223.

[5] J. Divyasree, H. Rajashekar, and K. Varghese, "Dynamically reconfigurable regular expression matching architecture," in *ASAP '08*, 2008, pp. 120–125.

[6] M. J. Wirthlin, "Constant coefficient multiplication using look-up tables," *Journal of VLSI Signal Processing*, vol. 36, no. 1, pp. 7–15, 2004.

[7] J.-L. Brelet and B. New, *XAPP203: Designing Flexible, Fast CAMs with Virtex Family FPGAs*, Xilinx, 1999.

[8] *ABC: A System for Sequential Synthesis and Verification*, Berkeley Logic Synthesis and Verification Group. [Online]. Available: http://www.eecs.berkeley.edu/ alanmi/abc/

[9] Xilinx, *DS586: LogiCORE IP XPS HWICAP*, Xilinx, 2010.

[10] ——, *UG071: Virtex-4 FPGA Configuration User Guide*, Xilinx, 2009.

[11] ——, *UG081: Microblaze processor reference guide(v9.2)*, Xilinx, 2008.

[12] ——, *UG070: Virtex-4 FPGA User Guide*, Xilinx, 2008.