Configuration Cloning: Exploiting Regularity in Dynamic DSP Architectures

S.R. Park W. Burleson

Dept. of ECE, University of Massachusetts
Amherst, MA 01003
{srpark, burleson}@ecs.umass.edu

Abstract

Existing FPGAs have fairly simple and inefficient configuration mechanisms due to the relative infrequency of reconfiguration. However a large class of dynamically configurable architectures for DSP and communications can benefit from special-purpose configuration mechanisms which allow significant savings in configuration speed, power and memory. Light weight configuration mechanisms allow much finer grained dynamic reconfiguration techniques of DSP and communications functions that tune algorithm and architecture parameters incrementally to track data and environment variations. These adaptive techniques exploit the time-varying nature of many DSP applications and avoid the power costs associated with worst-case design.

In this paper we develop a new FPGA configuration method called *configuration cloning* from the analogy of biological cloning. The main motivation behind configuration cloning is to exploit spatial and temporal regularity and locality in algorithms and architectures by copying and operating on the configuration bit-stream already resident in an FPGA. This results in a speed and power improvement over off-chip partial reconfiguration techniques but does require additional interconnects and control hardware on the FPGA. But in contrast to multiple-context and striped configuration approaches, cloning requires only a small amount of hardware overhead to greatly increase the on-chip configuration bandwidth. Details of the configuration cloning mechanism are described. Two familiar DSP applications, motion estimation and FIR filtering, are explained to demonstrate order of magnitude reductions in configuration time and power compared to existing configuration techniques. Resource recycling is also presented as a generic method of reclaiming freed up logic resources and using them as local memory to greatly reduce I/O requirements.

1 INTRODUCTION

Reconfigurable and custom computing is an emerging research area that has relied largely on the improving capacity and performance of commercial-off-the-shelf (COTS) Field Programmable Gate Arrays(FPGAs). Commercially available FPGAs provide flexibility and rapid prototyping to a wide variety of users, but their general applicability comes at the expense of relatively low logic density, low speed, high power consumption and cumbersome configuration. Dynamic reconfiguration is one of the most attractive possibilities of reconfigurable computing in which the FPGAs can be reconfigured to support different tasks in different time slots (time-multiplexing) or to adapt an algorithm/architecture to changing input signals and environments. The configuration mechanism is clearly one of the most important factors in dynamic reconfiguration since it determines the overhead and hence granularity of configuration. However the most widely used methods in commercial FPGAs, both *full configuration* and *partial configuration* do not exploit the inherent regularity and locality in many DSP and communications applications. And perhaps more importantly, the increasing size of FPGAs leads to ever increasing time, power and storage requirements for both initial and modified configurations. For example, one of the biggest FPGAs in the Xilinx XC4000 series[3], the XC4085XL, takes almost 2 seconds to configure the whole chip with a 2Mbit bit-stream. Several novel configuration methods have been developed[7][8][23] to locally store the configuration memory however they typically involve very large hardware overhead. If one thinks of configuration memory as the program memory for a configurable architecture, there are a plethora of more sophisticated architectural techniques from advanced computer architecture which can be employed to reduce configuration overhead. In this paper, we explore one of the most obvious which is the implementation of dynamically bound iterative constructs (loops in programs or regular arrays in architectures).

To implement regular arrays and dynamically modify the bounds of the arrays, we develop a new configuration method named *configuration cloning*, which comes from the timely analogy of biological cloning. A clone is an individual grown from a single body cell of its parent and genetically identical to the parent and cloning is the process of making a clone. Cloning provides a relatively simple mechanism to *increase* the length of various dimensions (e.g. word length, search space, window size, filter length, crypto key length, pipeline depth and queue length,...). in an array architecture to accommodate adaptive algorithms. Cloning is implemented by copying the bit-stream from one region of an FPGA to one or several other regions. In some cases this only involves modifying the logic and not the interconnect or vice-versa, hence we make a distinction between different

types of configuration bits. We stretch the biological analogy a bit by allowing some variations between the parent and the clone to accommodate irregularity. We also develop a dual technique to accommodate the dynamic *decreasing* of various array parameters. This technique is called *resource recycling* and allows resources to be disconnected from the operational portion of the array architecture and potentially used as a local memory to avoid off-chip I/O. This technique results in significant power savings by avoid unnecessary computation as well as reducing power-hungry off-chip I/O.

The rest of this paper is organized as follows. In section 2, configuration cloning and its implementation are explained in detail. In section 3, different types of configuration methods are explained and compared. Three example applications are explained in section 4 to show the usefulness of this method. A performance comparison is given in section 5 and finally conclusions and future work are described in section 6.

2 CONFIGURATION CLONING

As mentioned earlier, *configuration cloning* exploits regularity and locality in algorithms and architectures in the initial configuration as well as later reconfigurations. The configuration overhead here includes: the configuration time, configuration bit storage and power consumed in configuration. Cloning is best suited for array type applications with a host processor to coordinate configuration. The adaptation algorithm runs on the host hence it makes sense for the host to coordinate configuration. Since many signal or image processing algorithms can be implemented in array architectures, this method can be applied to a broad range of applications. In array type applications, an array is composed of processing elements(PEs) with interconnection wires among them. By using the cloning method, the overhead of the initial loading of configuration bit-stream can be significantly reduced. In some cases, which will be explained later, it is desirable to adjust the size of array processor to support different hardware parameters or to adapt the system to changing input signals. If the FPGA in use can not support partial configuration, the whole FPGA needs to be reprogrammed. In this case, the reconfiguration time and power can be large enough to cancel out the wins of small reconfigurations. Partially reconfigurable FPGAs are a step towards alleviating this problem.

Configuration cloning is particularly well-suited for this case. By copying the bit-stream already resident in an FPGA to multiple locations and using the configuration bit-stream lines simultaneously, cloning method can do the job in less time with less power consumption. In the XC6200 series, loading configuration bits to multiple destination is implemented using wild card [2]. While the wild card in the XC6200 series can be thought of as SCMD (Single Configuration Multiple Destination), cloning extends this to MCMD (Multiple Configuration Multiple Destination). This method is also scalable as FPGA becomes larger and larger. Fig.1 shows two reconfigurable architectures: a shared memory co-processor and a VLIW with an FPGA execution unit.

Fig.2 shows the inside of a tiny FPGA to explain the proposed configuration method. We suppose that the FPGA has an architecture similar to the Xilinx XC4000 series. The CLBs can be used as RAM if not used for implementing logic. A CLB in a cell consists of two 4-input look-up tables(LUTs). For simplicity, we assume that the FPGA does not have long interconnect wires which can be found in the

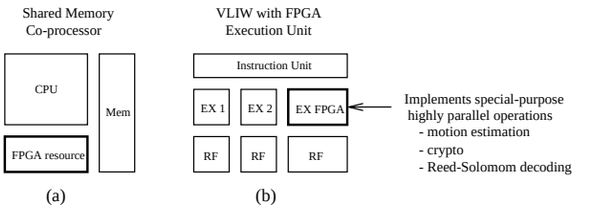

Figure 1: Two Configurable Architectures

XC4000 series. The whole *array* in Fig.2 is composed of 8 by 8 cells. A *cell* is composed of a CLB and accompanying interconnection wires. Cells are grouped into 2 by 2 to form a *subarray*.

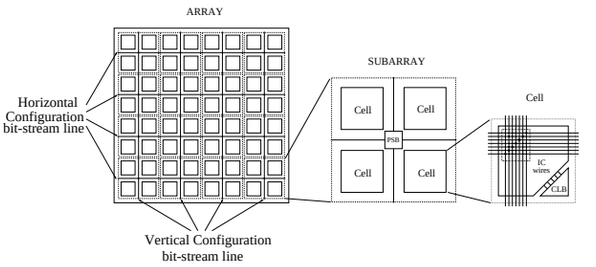

Figure 2: Inside the FPGA

Each subarray shares vertical and horizontal *configuration bit lines* which across the whole chip length to send or receive the configuration bit-stream data. The receiver and sender address sent by the command interpreter set the switches in Programmable Switch Box(PSB) in a subarray with the help of control circuit. Each subarray has its own address (e.g. C0R1 for Column 0 and Row 1) and each cell in a subarray can be distinguished. Furthermore the configuration bits for CLB and for interconnect in a cell can be distinguished. A host processor can load or copy configuration bit-stream from one location to another by simple commands. The commands are, essentially, binary data from a host processor to a command interpreter in an FPGA. Here we explain them with an assembly language like format for the sake of simplicity. The commands look like:

```
load destination, configuration data
copy direction of movement,
     (multiple)destination, (multiple)source
```

The `load` command loads the configuration bit-stream in a specific location. Fig.3 shows the details of the destination. By using a wild card similar to that of the XC6200 series[2], multiple destinations can be specified.

Subarray Address(2)	Cell ID(2)	CLB or IC(1)	Wild Card(5)

Figure 3: Details of destination in a command

In the `copy` command, the `direction of movement` specifies the direction of the configuration bit-stream movement, which is either vertical or horizontal, but not both. Multiple destinations can be used to copy the bit-stream into several subarrays. When multiple sources and multiple destinations are specified, the bit-stream in each source is copied to multiple destinations along the specified direction.

The cloning procedure is explained in Fig.4. In this example all the components of the array have the same function (hence the same structure) and the same interconnection. (This is not realistic since the components at the edges of a array can have different structure than that of the core. This limited irregularity can be easily handled with more commands.) The commands from a host processor are:

```
load C0R0, configuration bit-stream
copy horizontal, C1 C2 C3, C0R0
copy vertical, R0 R1 R2 R3, C0R0 C1R1 C2R0 C3R0
```

Fig.4(a), (b) and (c) show the configuration states after the execution of each command above. In the first command, C0R0 is the address of the lower left subarray. The configuration data from a host processor is loaded to C0R0. In the second command, the configuration bits in C0R0 are copied to the subarrays in the same row. Notice that the copy is done not one by one but simultaneously. In the third command, a multiple copy is done in the vertical direction.

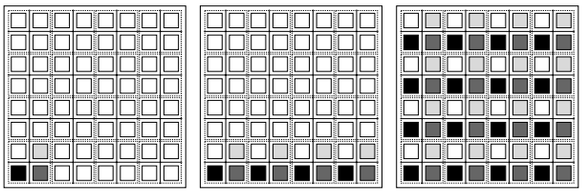

Figure 4: Example of configuration cloning procedure (a)initial loading of bit-stream (b)horizontal copy (c)vertical copy

More details of the configuration cloning operation are explained in the following. There is a simple command interpreter in the FPGA as shown in Fig.5. The command interpreter decodes the command from a host processor and broadcasts sender and receiver address to a proper configuration bit-stream line. To start configuration, a host processor writes the command into the command register in the command interpreter in the FPGA. If the command is `load`, the command interpreter selects the proper configuration line and sends destination address to indicate the destination. After this, the configuration bit-stream is sent along the configuration bit line. Only the activated destination receives(updates) the configuration bit-stream. In the case of a `copy` command, the command interpreter indicates(activates) the source(s) and destination(s). After this, the source sends the configuration bit-stream along the configuration bit-stream lines and the destination(s) receives the bit-stream.

Fig.6 shows the detail of a subarray to support configuration cloning. All subarrays in a row(column) are connected to the command interpreter via a configuration bit-stream line as shown in Fig.5(b). The command interpreter can specify the source and destination subarray, cell or CLB/IC by setting the switches in the PSB in subarrays. The setting of the switches is done by the control circuit in a subarrary(not shown in Fig.6). Fig.6 shows the configuration-related parts in a subarray. The configuration memory in a subarray is divided into 8 chunks of scan chains. The basic units of configuration are configuration for CLB and for interconnect(IC). The programmable switch boxes and programmable switches between CLBs and ICs are set according to the source and destination specified by the command

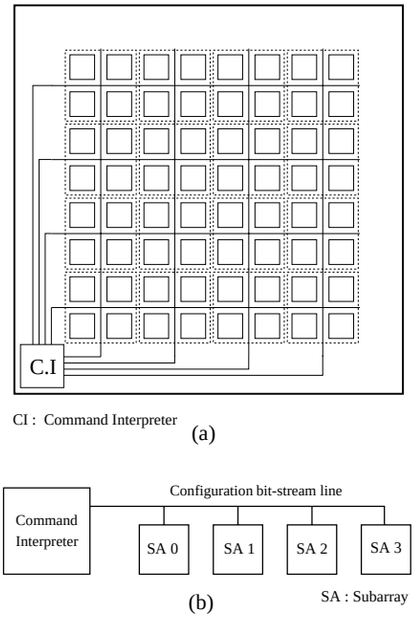

Figure 5: Command Interpreter in a FPGA

from the host. After all source and destinations are set, the configuration bits move via the configuration bit-stream line by the configuration clock. The number of clocks necessary is set in the counter in the command interpreter, since it can be known from the command from the host.

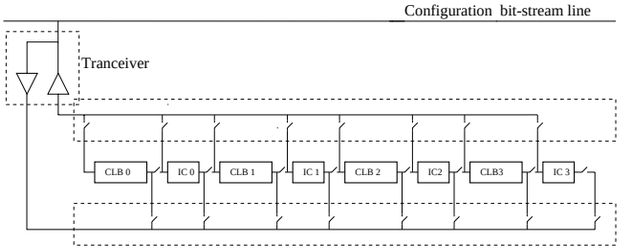

Figure 6: Programmable Switches in a Subarray

3 DYNAMIC CONFIGURATION METHODS IN FPGAS

Configuration in an FPGA is the process of loading design-specific configuration data into an FPGA to define the function of logic blocks and their interconnection. A large class of dynamic reconfiguration applications simply multiplex the same logic over a group of tasks (e.g. coarse-grained multiplexing in video coding[17], swapped crypto/compression coprocessor for wireless LAN[14] fine grained multiplexing of hardware(Cache logic[6]). Although this class of applications is challenging, we are not addressing it in this paper since 1) it can usually be done statically, and 2) we feel that the ever increasing size of FPGAs will make it an increasingly less necessary technique. Instead, we explore truly adaptive systems that can use dynamic configuration to track statistical variations in the computations. These occur at a wide range of time scales(nanosecond, microsecond, millisecond, second) and can also occur within

or between tasks. For example an environmental parameter like temperature could be hard-wired on a very slow (seconds) basis due to the slow rate of change even in an unmanned aircraft radar[13]. A faster configuration might occur at frame-rate (milliseconds) due to the changing content in a video sequence[11]. Micro-second level reconfiguration would be useful for a mobile radio network channel due to changing channel characteristics and network traffic [25]. Finally, nano-second level reconfiguration can be used to rapidly change architectural features like register sizes and pipeline depth to match variations in computational properties.

In addition to these varying time-scales, we also define four different types of dynamic configuration which have quite different requirements:

- *Algorithm level configuration*
  Agile crypto[22], Environment-dependent radar[13]
- *Algorithm/Parameters configuration*
  Motion estimation[11], Modem equalizer parameters [26]
- *Architectural level configuration*
  DISC[15]
- *Hard-wiring*
  Automatic Target Recognition[12], Hard-wired filters [19], Hard-wired crypto keys[21]

In the following section, several different types of configuration methods are explained briefly and matched to the list above.

- **Full Configuration**
  The first and most typical configuration method for FPGAs is *full configuration*. The entire FPGA should be programmed before the beginning of the operation. To a user, the configuration process looks like the loading of a long serial configuration bit-stream to the configuration memory in an FPGA. The configuration time is usually on the order of 10's of msec and is growing linearly with the ever growing size of FPGAs. Widening the serial path to 8, 16, 32 bits improves the time, but does not improve power. Due its slow speed, full configuration is limited to very slow adaptations on the order of milliseconds.
- **Partial Configuration**
  More recently developed FPGAs such as the XC6200 series from Xilinx [2], CLAy from National Semiconductor [18], ORCA series from Lucent Technologies[4] and AT6000 series from ATMEL[6] support *partial configuration*. A part of the chip can be reconfigured while another part is operating. The XC6200 series also has more features than just partial configuration, such as to write the same configuration data to several cells and to interface a host processor directly[2]. Partial configuration can be applied efficiently down to the nanosecond level but pays a price in extra hardware.
- **Multiple-context Configuration**
  A DPGA has multiple-context configuration bits and a context switch can be achieved by broadcasting a global context identifier[8]. A context switch can be done in a clock cycle, but at an extremely high H/W cost.
- **Dynamic Compilation**
  The work by Mangione-Smith mentioned in [20] uses small units of precompiled FPGA configuration bit-streams to overcome the disadvantage of slow FPGA place and route tools. Configuration bit-streams are combined very quickly at run time to constitute a full configuration bit-stream. This is more of a fast place and route technique, but could also be used as part of a partial reconfiguration approach. It seems to target complex reconfigurations that require re-running of CAD tools in real-time and thus is limited to the milli-second level or above.
- **Striped Configuration**
  Proposed in [7], striped configuration is a method of partial configuration that is well suited for pipelined applications.
- **Configuration Cloning**
  In our proposed configuration method, portions of the configuration bit-stream can be copied from one location to another in time proportional to the size of the partial bit-stream (depending on the length of the partial bit-stream this could be nanoseconds up to seconds). The main advantage of cloning is avoiding the high cost of off-chip communication through straightforward compression and incremental generation of regular bit-streams. Multiple destinations can be specified to reduce the configuration time in array applications. Cloning was conceived for applications which use dynamic configuration to incrementally tune algorithm/architecture parameters to track data variations (e.g. word length, search space, window size, filter length, crypto key length, pipeline depth and queue length,...).

The pros and cons of the above configuration methods are shown in Table.1. The first two methods: full configuration and partial configuration are targeted to general purpose applications. The last three methods are more application specific. It is our belief that the best configuration method depends highly on the application area, but that if the application area is large enough, new FPGA hardware and software are warranted. Cloning presents another approach worth considering for a large class of applications with regular structures that vary dynamically.

4 EXAMPLE APPLICATIONS

Three example applications of configuration cloning are explained in this section. The first one is motion estimation for video coding implemented in a two dimensional systolic array. Cloning is a good match for this example due to its two dimensional regularity and easily routed array structure. It is also the example that motivated our interest in cloning to avoid the time and power costs of very fine-grained reconfiguration. We then explore the FIR filter and notice that a canonical implementation that preserves tap regularity is preferred for cloning over an efficient distributed arithmetic implementation. We then present a generic technique for generating dynamically variable-sized local memories using recycled resources and illustrate this in the motion estimation example.

4.1 Motion Estimation

Motion estimation is the most computationally demanding part of video coding and widely used in video coding stan-

config. method	*main features*	*best suited*	*drawbacks*	*H/W support*
full config.	simple config.	fast prototyping	long config. time	scan chain
partial config.	selective config.	fast reconfig.	additional H/W overhead	random access
multiple context config.	fast swapping of multiple context	time multiplexable tasks	more application specific	multiple copies of config. memory
striped config.	pipeline stage level config.	pipelined tasks		wide on-chip config. cache
config. cloning	incremental tuning of algo/archi parameters	array type applications	H/W overhead	extra routing and circuit per subarray

Table 1: Comparisons of Configuration Methods

dards such as H.261, MPEG-1, MPEG-2 and even MPEG-4. For estimating motion by means of a block matching algorithm, the image is divided into blocks of $n \times n$ pixels. Usually n is 16. The blocks resulting from the segmentation of the current and previous frames are called the current and previous block, respectively. For each current block, the best matching previous block is found within a search area surrounding the previous block(Fig.7(b)). The previous blocks in a search area are called candidate blocks. Suppose the search area extends on both sides over p pixels in the horizontal and vertical directions, then the search area is $(2p+n)^2$, and the total number of the candidate blocks in search area is $(2p+1)^2$. The picture size is M by N pixels. Fig.7 shows a block and the block matching algorithm.

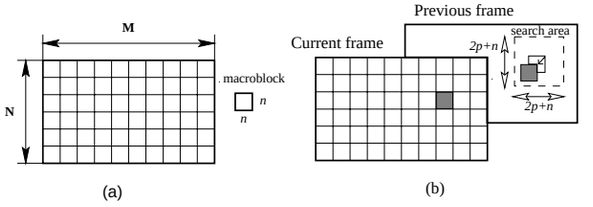

Figure 7: (a) Block in a frame (b) Motion Estimation by block matching

To compute the motion vector, the Mean Absolute Difference (MAD) criterion is widely used.

$$D(k,l) = \sum_i \sum_j |x(i,j) - y(i+k, j+l)|, \qquad (1)$$

where $x(i,j)$ is the luminance value of a pixel in the current block and $y(i+k, j+l)$ is the luminance value of a pixel in the candidate block. V_{min} is called the motion vector. The displacement calculated is limited to a search area range such that $-p \leq k, l \leq p$.

The two dimensional systolic implementation is shown in Fig.8 [10]. The 'AD block' computes the absolute value and summation and the 'R block' is composed of a mux and registers. The '+ block' accumulates the results and the 'M block' determines the V_{min}.

The statistics of motion vectors vary considerably between and within video sequences. Fig.9 shows the distributions of the horizontal component of the motion vector of 'Miss America' and 'table tennis' video sequences. The first 100 frames were used for collecting the motion vectors. As can be seen, the range and the shape of the distribution of

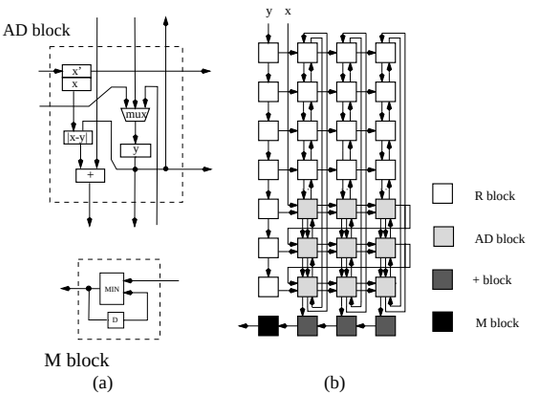

Figure 8: practical implementation[10] (a) details of AD block and M block (b) architecture ($n = 3, p = 2$)

motion vectors differ due to the different types of motion in the two sequences. In the 'table tennis' sequence, even in the same sequence, the shape of the motion vector distribution varies due to the the changing characteristics of the image.

If there is little motion in the picture, the search range can be reduced without sacrificing the picture quality. The result is substantial power saving by avoiding unnecessary computation.

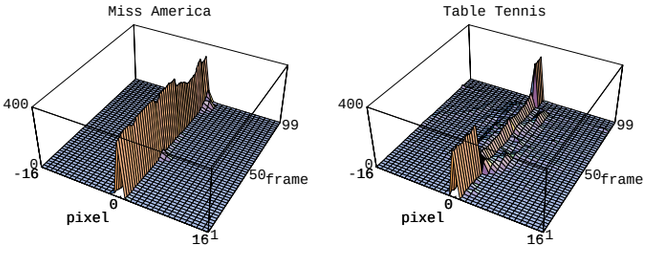

Figure 9: Motion Vector Distribution over time in 'Miss America' and 'table tennis' video sequences.

Fig.10 (b) is a simplified representation of Fig.8 (b). Fig.10 (c) is a reconfigured architecture to support a larger search area than Fig.10(b). The cloning method can exploit the regularity in this array architecture. To increase the array size from Fig.10(b) to (c), two steps of the copy com-

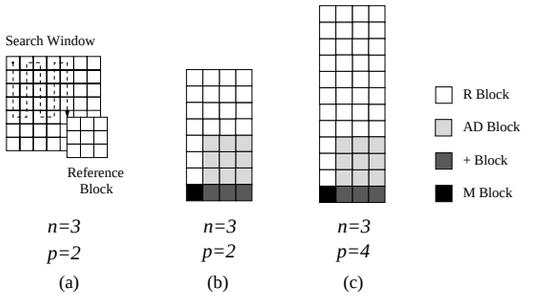

Figure 10: (a) Kernel of the search algorithm[10] (b) simplified block diagram ($n = 3, p = 2$) (c) reconfigured architecture to support a larger search area ($n = 3, p = 4$)

mand are sufficient. First, the uppermost row in Fig.10(b) is copied to the desired location and then the next row is multiple copied. Notice that the configuration time does not depend on the amount of reconfiguration in this example. Limited irregularity is also well handled. Moreover the cloning method outperforms partial configuration, since it can exploit the wide configuration bit lines simultaneously and copy the configuration bit-stream to multiple destinations at the same time.

4.2 FIR Filter

Filtering is the most fundamental of DSP algorithms. Digital filters can be implemented in many ways. Programmable DSPs lie at one end of the spectrum and dedicated ASICs the other end. FPGAs can also efficiently implement digital filters and several recent efforts have explored reconfigurable implementations [9] [19]. A key idea in reconfigurable filters is the distinction between the configuration of the filter coefficients (which has no regularity) versus the configuration of the filter length and word lengths for both data and coefficients (usually results in regular interconnections). This idea also emerges in cryptography keys [21].

The Finite Impulse Response(FIR) filtering can be expressed as follows.

$$y[i] = \sum_{k} x[k-i]c[k] \qquad (2)$$

where $c[i]$ is filter coefficients and $x[i]$ is input data.

Fig.11 shows a typical implementation of an FIR filter. As can be seen, the FIR filter has a regular structure, hence can be easily cloned. Different numbers of taps for different input signal statistics can be accommodated easily.

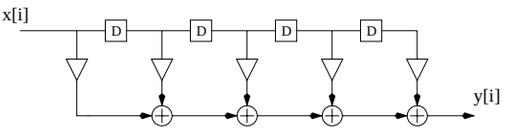

Figure 11: Typical Implementation of FIR Filter

Fig.12 shows the 16-tap FIR filter implementation using LUT-based Serial Distributed Arithmetic[9], where the Multiply and Accumulation(MAC) of input data and filter coefficients is implemented by Look-up tables(LUTs). The filtering operation is done in a bit-serial way. Although this implementation is more practical in LUT-based FPGAs, the

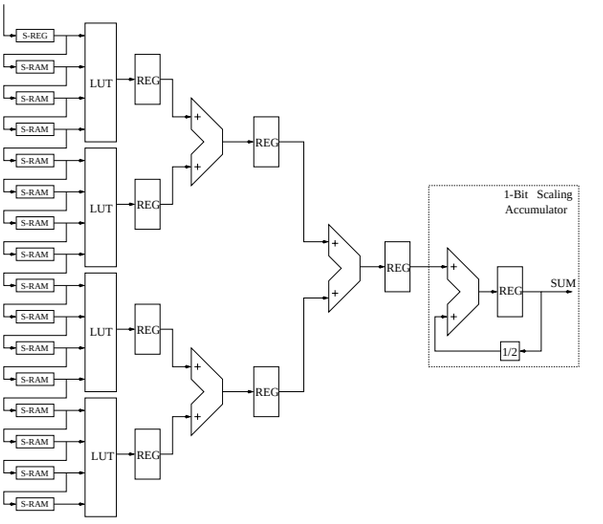

Figure 12: 16-Tap FIR Filter using LUT-based Serial Distributed arithmetic[9]

structure is not regular at the tap-level, hence can not be easily cloned. However cloning could be accommodated in the coefficient word length dimension by cloning an additional slice onto all datapaths and then loading new values into the distributed arithmetic lookup tables.

4.3 Local Memory Structure with Resource Recycling

A very useful application of cloning occurs in the implementation of variable length buffers and queues. In many signal processing algorithms, a significant number of redundant off-chip memory accesses are required because data-structure are simply too large to fit in on-chip memory. This problem is particularly acute in FPGAs where memory is fairly expensive. Cloning can be used to efficiently instantiate arrays of special-purpose memories. In addition, the size of those memories can then be adjusted with the cloning mechanism. In some cases, a large fraction of the unused resources on the FPGA can be *recycled* and used as memory. Memories like queues and FIFOs are fairly easy to route and can be partitioned to fill up much of the space in an otherwise fractured floor plan. The data structures in image and video signal processing algorithms and presumably many other applications are currently too large to fit on FPGA chips. So rather than rely on generic cache mechanisms which are rarely supported in commodity FPGAs, our techniques allow an explicit control of the memory structures which are quite predictable and structured in signal processing. Most commercial FPGAs provide an alternative use of the CLB configuration memory as local memory. This can be combined with the CLB registers to build small distributed memories.

An example of this occurs in our reconfigurable motion estimation system described in [11]. When the parameter of an array architecture is reduced due to changes in the environment and reduced computational demand, the freed-up resources can be used as local memory to hold a piece of the data-structure that is repeatedly used by the algorithm. In the motion estimation case, a portion of the image will be used again to search for matching blocks in a two dimensional neighborhood of adjacent pixels(Fig.13).

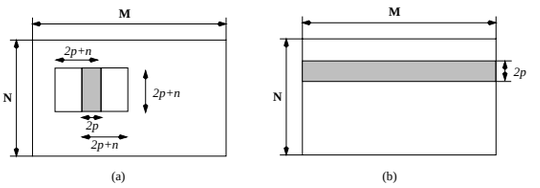

Figure 13: Data structure reuse in sliding window-based image processing (a) horizontal overlapped area (b) vertical overlapped area

The local memory reduces the off-chip I/O and hence can significantly reduce power consumption. Fig.14 shows an example that freed up resources are used as local memory. The following table shows the power savings due to reducing the search space size in our motion estimation example both with and without resource recycling for local memory.

search area(p)	P_{comp}	$P_{I/O}$	$P_{I/O}$ with recycled local memory
16	1.00	1.00	1.00
14	0.77	0.86	0.81
12	0.57	0.73	0.63
10	0.40	0.61	0.46

Table 2: Power Saving with recycled local memory

Each term was normalized to 1 when $p = 16$. The picture size is 720×576(CCIR 601 format) and $n = 16$. When estimating the $P_{I/O}$ with local memory, we assumed that we build local memory only by returning resources from reducing the search area. Returning resources are used for local memory for horizontal overlapped area(Fig.13(a)). How much local memory can be built from returning resources depends on the FPGAs being used. Our estimates are based on on the Xilinx XC6264.

By reducing p from 16 to 10, P_{comp} was reduced by 60 % and $P_{I/O}$ with local memory by 54 %. The effect of utilizing unused or returning resources as local memory is larger than one may expect due to the very high cost of off-chip I/O. When reducing p from 16 to 14, $P_{I/O}$ reduced by 14 % but with local memory total 19 % of reduction in $P_{I/O}$ can be obtained.

Reduced memory accesses can also have other benefits (e.g. reduced conflicts on an external shared bus, reduced use and pollution of an external shared cache, etc.).

Power savings through resource recycling provides an answer to the bothersome question of what to do with unused resources once you have down-sized your computational structures. Occasionally it will be possible to find algorithms in which one dimension decreases while another increases (for example number of filter taps and word length) resulting in a fixed overall amount of hardware. However we feel that this is a rare case and resource recycling for local memories provides a much more generic technique. Although we don't illustrate further examples, we expect resource recycling is a fairly generic technique in reconfigurable DSP and other algorithms which can exploit dynamically variable memory sizes. This would include queues, buffers, dictionaries and other large data structures. It is an architectural low-power technique that requires some hardware support but probably not more than that required for sub-system power-down or dynamic supply voltage scaling [24].

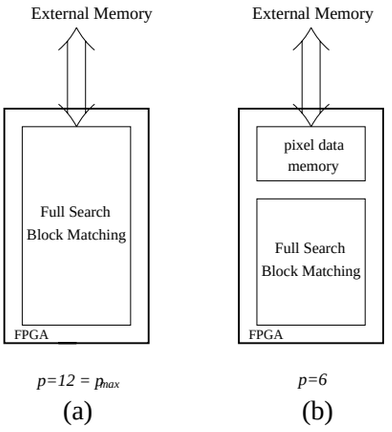

Figure 14: Resource Recycling in Motion Estimation

5 PERFORMANCE COMPARISON

The amount of impact of reconfiguration to overall system performance depends on applications. For example, when FPGAs are used for fast prototyping and are configured only once at the beginning of the operation, slow full configuration may not be a problem. For real-time reconfigurable DSP, slow reconfiguration can result in most operation time being wasted for reconfigurations[19].

First we look at the configuration time. As we mentioned earlier, the best configuration method depends strongly on the application. Therefore we do not try to compare configuration cloning to all configuration methods in section 3. Instead we compare our method to full configuration and partial configuration. To make the comparison fair, we suppose the following. Basic architectures of three FPGAs: full configuration, partial configuration and configuration clone are the same and only the configuration methods, hence the additional hardware to support them, are different. All three methods have the same configuration clock speed, although configuration cloning can have a higher clock rate since it does not rely on off-chip I/O. By partial configuration, we mean the ability to configure a part of an FPGA independently of other parts. Furthermore we suppose that all configurations in the three methods are done in a bit-serial way. For some commercial FPGAs such as the XC6200 series have more features than a simple meaning of partial reconfiguration (e.g. wild card register, FastMap register, direct interface to microprocessor etc). We do not consider them in the comparison for two reasons. One is these features are more like other aspects of configuration than a simple meaning of partial configuration. The other is to support these features, cost, silicon area, should be paid.

We use the motion estimation example in section 4 for the comparisons. Since the configuration time is proportional to the number of configuration bits, we count the number of configuration bits to measure the configuration time. we consider the following situation in motion estimation. We want to increase the search space to accommodate a large motion in a picture, which can occur after a scene change. The search space will be adjusted from $p = 4$ to $p = 8$, and n is fixed to 16 as in most video compression standards. From the Fig.10 (b) and (c), we know that 8 rows of 'R block' are to be added.

For each configuration method, the configuration time is:

- *Full Configuration*

the total number of cells × configuration time for one cell

- *Partial Configuration*
  the number of cells to be reconfigured × configuration time for one cell
- *Configuration Cloning*
  the steps needed in reconfiguration × configuration time for each step

We assume that 44 by 44 array of cells is necessary to implement 1 bit RBMAD[16] motion estimation, and that the 'R block' in Fig.10(b) takes up 1 cell. The numbers of cells to be reconfigured and the number of steps needed in each configuration method are:

full configuration : 44 * 44 = 1936 cells
partial configuration : 16 * 2 * 4 = 128 cells
configuration cloning : 1 + 1 = 2 steps

Since the time for one step in configuration cloning is almost the same as for reconfiguring one cell, we can observe that significant reduction in reconfiguration time can be achieved by reusing the configuration bit-stream.

An additional advantage of the cloning method is for saving power during the configuration. This occurs in at least four ways:

1. Cloning can reduce power-hungry off-chip I/O by reusing the configuration bits already resident on the FPGA.
2. Even if the bit-stream comes from off-chip, cloning allows a more compressed bit-stream due to its exploitation of regularity.
3. By avoiding using a single large configuration bus, switching capacitance is reduced,
4. Cloning also wins over the traditional method of using a small number of serial bit-streams for configuration, due to reduced switching capacitance and improved signal correlations that can result in fewer node transitions,

The configuration time and power of the motion estimation example are summarized in the following Table. For configuration power comparison, we assume that the power is proportional to the number of cells to be reconfigured and the capacitance of off-chip I/O and a configuration bit-stream line are 50pF and 1pF, respectively. Note that this uses a simple model of dynamic CMOS power dissipation and does not model transition probabilities or static power.

	time	power
full configuration	1936	1936
partial configuration	128	128
configuration cloning	2	0.44

Table 3: Comparisons of Configuration Time and Power

6 CONCLUSIONS AND FURTHER WORK

A new configuration method called *configuration cloning* was proposed and the details were explained. Unlike many commercial FPGAs, configuration cloning can reuse the configuration bit-stream already loaded in an FPGA. This can greatly reduce the configuration overhead in initial loading of configuration bit-stream and later fine tuning of algorithm/architecture to adapt the system to input data. More than an order of magnitude of configuration time win over full configuration can be obtained in some applications at some cost of silicon area. Two common examples in image and signal processing areas, motion estimation and FIR filtering, are explained. Cloning works well for motion estimation due to its preservation of regularity while it is not a good match for the FIR filter due to the shared distributed arithmetic look-up tables across multiple filter taps. In addition, the capability of recycling unused resources into memory with the help of cloning can help increase on-chip memory thus avoiding power hungry off-chip I/O. A prototype cloning FPGA is currently being designed in .25 μ CMOS to verify the area, speed and power overhead of the special-purpose cloning hardware.

Further work in this area includes:

1. CAD tools are needed to support cloning. Exploiting and preserving regularity is not a priority in current FPGA place-and-route tools, therefore new approaches are needed. We do not expect that clonable designs will be automatically identified from a high-level description of automatically generated. We expect that some hand-design will be needed to develop clonable macros but that these could then be composed with automated methods. Despite numerous efforts, automated methods for identifying and exploiting regularity in general-purpose computation remain an elusive goal. Techniques also need to be developed which allow clonable portions of the circuits to exist with more irregular non-clonable circuitry.
2. FPGA architectures need to be developed which efficiently implement clonable macros as well as enabling the actual cloning mechanism. This involves supporting regular structures.
3. A run-time operating system needs to orchestrate cloning by providing some external bit-streams as well as controlling the copying and manipulation of bit-streams on chip. It would also coordinate resource recycling drawing on techniques from garbage collection.
4. Techniques for supporting minor irregularities would also be useful. For example, if the only thing that varied across the slices of a particular array was the reset polarity of a single register, it should be fairly easy to modify this in the configuration without requiring a full bit-stream.
5. Configuration cloning shares some ideas with the dynamic compilation suggested by Mangione-Smith [20] except that we combine bit-streams at a much later stage of the configuration process (ie on-chip). There are obvious limitations to cloning if the compiled bit-streams are highly optimized and hence difficult to combine. But if regular computations can map to regular bit-streams rather than highly optimized and flat bit-streams, it should be possible to merge and modify bit-streams on-chip.

6. More applications need to be explored to see if this technique has general applicability. We realize that we have tuned it highly to our Motion Estimation example from which it emerged. However we have strived to keep the techniques generic.

7. More work is needed on resource recycling. This technique could be quite powerful and draw on the large literature in distributed memory.

8. Regular routing resources can be used to route the configuration bit-stream. This obviously requires care to avoid reconfiguring the configuration mechanism circuitry during a configuration.

9. Quantitative comparison of configuration time and power for FIR filters and other applications with programmable coefficients is needed.

References

[1] B.L. Hutchings and M.J. Wirthlin, "Implementation Approaches for Reconfigurable Logic Application", FPL'95, pp.419-428,Oxford,1995.

[2] Xilinx, XC6200 Field Programmable Gate Arrays Data Sheet, Ver. 1.8, 1996.

[3] Xilinx, XC4000E and XC4000X series Field Programmable Gate Arrays Data Sheet, Ver. 1.4, Nov. 1997.

[4] Lucent Technologies, ORCA OR2CxxA and OR2TxxA Series Field-Programmable Gate Array Data Sheet, Jan. 1998.

[6] ATMEL, AT6000/LV Series Data Sheet.

[7] H. Schmit, "Incremental Reconfiguration for Pipelined Applications", Proceeding of IEEE workshop on FPGAs for Custom Computing Machines, 1997.

[8] A. DeHon, "DPGA Utilization and Application", FPGA'96.

[9] G.R.Goslin, "A Guide to Using Field Programmable Gate Arrays(FPGAs) for Application-Specific Digital Signal Processing Performance", Proceedings of High-Speed computing, Digital Signal Processing and Filtering Using reconfigurable Logic, 1996, SPIE Vol.2914, pp.321-331.

[10] P. Pirsch, N. Demassieux, and W. Gehrke, "VLSI Architectures for Video Compression -A Survey," Proc. of IEEE, vol.83, pp.220-246, Feb, 1995.

[11] S.R. Park and W. Burleson, "Reconfiguration for Power Saving in Real-Time Motion Estimation", ICASSP, 1997.

[12] J. Villasenor, B. Schoner, et al, "Configurable computing solutions for automatic target recognition", Proceeding of IEEE workshop on FPGAs for Custom computing machines, pp.70-79,1996.

[13] M. Petronino, R. Bambha, J. Carswell, and W. Burleson, "An FPGA-based Data Acquisition systems for 95GHz W-band Radar", ICASSP, 1997.

[14] A. Brahmbhatt, and W. Burleson, "FPGA-based Coprocessors for Wireless Data Communications", Massachusetts Telecommunications Conference, 1997.

[15] M.J. Wirthlin, and B.L. Hutchings, "DISC: The dynamic instruction set computer", Field Programmable GateArrays (FPGAs) for Fast Board Development and Reconfigurable Computing, Proc. SPIE 2607, pp. 92-103, 1995.

[16] Y. Baek, H.-S. Oh, and H.-K. Lee, "An efficient block-matching criterion for motion estimation and its VLSI implementation", IEEE Trans. Consum. Elec. Vol.42, pp.885-892, Nov. 1996.

[17] B. Schoner, C. Jones, and J. Villasenor, "Issues in Wireless Video Coding using Run-time reconfigurable FPGAs", Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines, 1995.

[18] National Semiconductor, "Configurable Logic Array(CLAy) Data Sheet, 1993.

[19] M.J. Wirthlin and B.L. Hutchings, "Improving Functional Density Through Run-Time Constant Propagation", FPGA'97, 1997.

[20] J. Villasenor and B.L. Hutchings, "The Flexibility of Configurable Computing", IEEE Signal Processing Magazine, pp.67-84, Sep. 1998.

[21] J. Leonard and W.H. Mangione-Smith, "A Case Study of Partially Evaluated Hardware Circuit: Key-Specific DES", International Workshop on Field Programmable Logic and Applications, 1997.

[22] personal communication with C. Paar, http://ee.wpi.edu/People/faculty/cxp.html

[23] http://www.cs.berkeley.edu/projects/brass/

[24] J. Rabaey, "Digital Integrated Circuits", Prentice Hall, 1996.

[25] D. Goeckel, "Strongly Robust Adaptive Signaling for Time-Varying Channels," *Proceeding of the 1998 International Conference on Communications,* pp. 454-458, June 1998.

[26] M. Goel, N. Shanbhag, "Low-Power Equalizers for 51.84 MB/s Very High-Speed Digital Subscriber Loop (VDSL) Modems, IEEE Workshop on Signal Processing Systems, 1998.