Systems and Techniques for Fast FPGA Reconfiguration

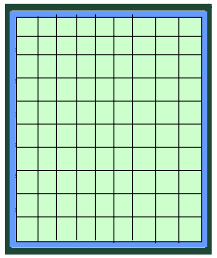
Usama Malik
School of Computer Science and Engineering
University of New South Wales
Sydney, Australia

The Thesis

- This thesis examines the problem of reducing reconfiguration time of an FPGA at its configuration memory level.

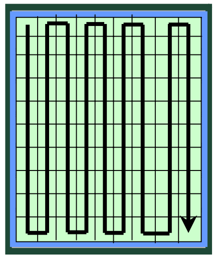
Existing Designs

- An SRAM-based FPGA consists of logic cells and switches that can be configured to realize an on-chip circuit
- The device is configured by loading configuration (or instruction) data in the configuration SRAM
  - The SRAM can be thought of as a local instruction cache
- Dynamic reconfiguration involves re-loading the configuration data in order to the change the behavior of the executing circuits
  - This corresponds to our cache misses in the general problem



Existing Designs

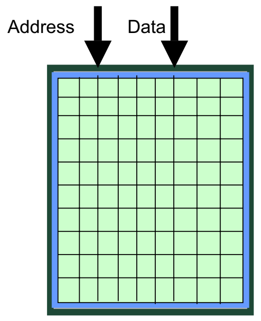
- **Various configuration distribution models exist**
  - A shift register solution (XV4000 series)
    - Synchronous update of the entire memory
    - Simple but constant reconfiguration delay



Existing Designs

Intro
Analysis
New Arch
Summary

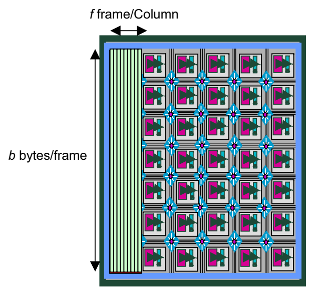
- **Various configuration distribution models exists**
  - A shift register solution (XV4000 series)
    - Constant reconfiguration delay
    - Synchronous update of the entire memory
  - RAM style addressing (XC6200 series)
    - Byte sized instructions
    - Synchronous update of *k* memory cells
    - Partial reconfiguration reduces the reconfiguration bandwidth
    - Scalability issues
      - » Significant on-chip wiring resources needed



Existing Designs

Intro
Analysis
New Arch
Summary

- **The Virtex Model**
  - Combines the shift register model with the RAM model
    - Synchronous update of a portion of a memory-column
  - Instruction size = 18X#rows+2x18 bits (more than 150bytes for a large device)
  - Single configuration port for address and data
    - Pin limitations in large devices
    - Reconfiguration time is proportional to the amount of frame data plus the address data
  - DMA style addressing
    - Load the address of the first frame and the number of consecutive frames to follow
- Our target device
  - State-of-the-art
  - Widely used in research
  - Have associated CAD tools available



Analyzing Partial Reconfigurablity in Virtex

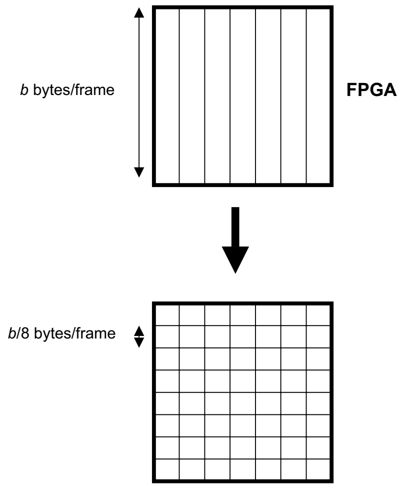
Intro
Analysis
New Arch
Summary

- **The configuration re-use problem**
- **Input:** A sequence of configurations
- **Aim:** To minimize the total number of frames to be loaded
- **Algorithm**
  - Place the first configuration on chip.
  - For each *next* configuration in the sequence
    - Load the frames that are present in the *next* but are different from the current on-chip frames at the same addresses.
- **Results**
  - For a sequence of thirteen benchmark circuits, 1% of the frames were re-used (Target device was an XCV1000).
  - A judicial placement of circuits to increase the amount of overlap between successive configurations increased the re-use to 3%.

The Effect of Frame Granularity

Intro
Analysis
New Arch
Summary

- Motivation
  - A single bit change in a frame can lead to loading the entire frame (156 bytes).
  - Break the frame into sub-frames and assume that each sub-frame can be independently loaded on the device.
- Results
  - At single byte granularity up to 78% of frame data was removed for the same circuits (assuming fixed placements)



The Configuration Addressing Problem

Intro
Analysis
New Arch
Summary

- Decreasing the size of the configuration unit can reduce the reconfiguration bandwidth requirements.
- However, increasing the number of configuration units increases the overhead in terms of address data.
- Assuming a RAM style addressing the overall reduction in the previous case was calculated to be 34%.
- Thus, the address data is a significant factor in consuming bandwidth motivating the need to study configuration addressing schemes.

The Configuration Addressing Problem

Intro
Analysis
New Arch
Summary

- Let there be *n* configuration registers in the device numbered from 1 to *n*.
- We are given an address set $\{a_1, a_2, a_3 \ldots a_k\}$ where $1 \le a_i \le n$, $1 \le k \le n$.
- Our goal is to find an efficient encoding of the address set
  - The address string must be small so that it demands less configuration bandwidth
  - The address decoding must be fast so that the decoder delay is small
- Next we study the run-length encoding (or the DMA model) of the address set.

The DMA Analysis

Intro
Analysis
New Arch
Summary

- The previous analysis was repeated for a set of ten benchmark circuits from the signal processing domain mapped onto an XCV100 device (90,160 bytes per complete configuration)
- The total amount of frame data under the available Virtex model was 684,944 bytes for a sequence of nine circuits (we assumed that the first circuit was already on-chip)
- DMA performed best at 2-byte granularity
  - 42% reduction in the amount of configuration data compared to the existing model
- Performs similar to the RAM model at single-byte granularity

Sub-Frame Size (B)	Sub-Frame Data (B)	RAM Address (B)	DMA Address (B)
8	390,725	83,290 (39%)	35,382 (41)
4	322,164	151,014 (31%)	76,819 (41%)
2	**248,620**	**248,620** (27%)	**144,104** (42%)
1	164,121	348,758 (25%)	365,211 (22%)

➡ A better configuration addressing scheme is required

The Vector Addressing (VA) Technique

Intro
Analysis
New Arch
Summary

- Unary or one-hot encoding of the address set
- Define a bit vector of size *n* bits where *n* is the number of configuration registers in the device
  - Set the i_{th} bit in the vector if the i_{th} register is to be updated else clear it to zero
- For the same sequence of circuits a maximum of 60% reduction in the configuration data was observed.

Frame Size (B)	#Frames in an XCV100	Total VA Data (B)	% reduction compared to current Virtex
8	11,270	12,679	41
4	22,540	25,358	48
2	45,080	50,715	51
1	90,160	101,430	60

Vector Addressing: Theoretical Considerations

Intro
Analysis
New Arch
Summary

- The VA method has a constant addressing overhead of *n* bits compared to the RAM method which gives $k\log_2(n)$ bits
- Compare $n < k\log_2(n)$
  - VA method is better than the RAM method as long as $k > n/\log_2(n)$
    - This has been shown for *core* style reconfiguration where an entire circuit is swapped with an other (e.g. a filter by an encryption circuit).
- Another use of dynamic reconfiguration is making a small update to the on-chip circuits (e.g. updating filter coefficients)
  - The above inequality is not likely to be true in these case
- In order to cater for the needs of reconfiguration at opposing ends of granularity combine DMA with VA
  - Enhance the current Virtex Model by incorporating the VA at the frame level

Deriving the New Memory Architecture

Intro
Analysis
New Arch
Summary

- Consider RAM style implementation of DMA-VA
  - Frame registers implemented as a column of independent registers
  - A frame address decoder selects a column (i.e. a frame)
  - Add a vector address decoder (VAD) that selects a row
- Problem
  - Too many wires
- Consider a read-modify-write strategy
  - In Virtex frames are first written in an intermediate buffer called *frame data register* (FDR) and then shifted in their final destination
- Read a frame into FDR, modify it and write it back
  - Keeps the shift register implementation of frame registers intact
- Problem
  - The bandwidth mismatch
  - Frames must be read/written fast enough otherwise the benefit of partial updates will be lost

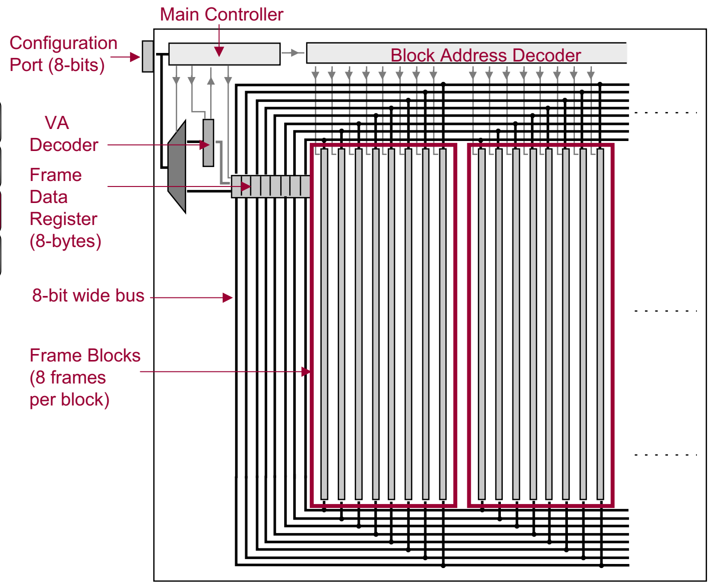
Deriving the New Architecture

Intro
Analysis
New Arch
Summary

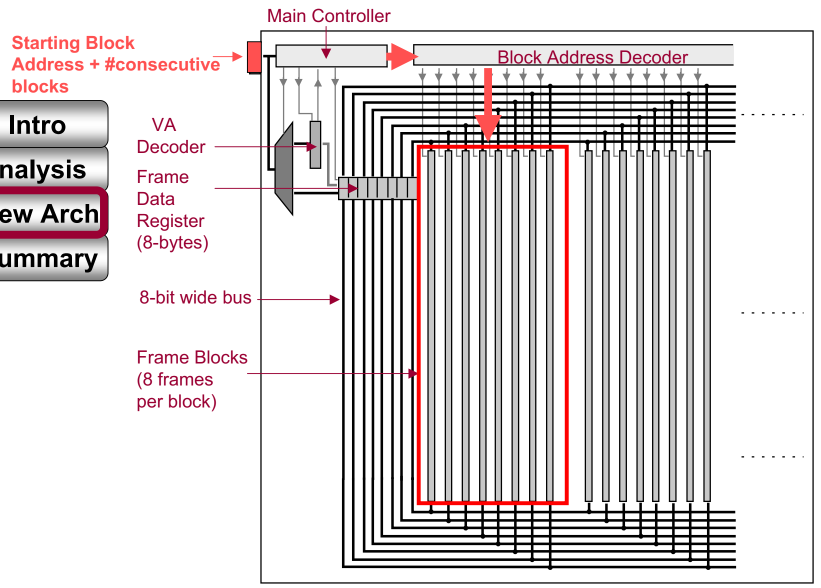
- Let the configuration port be of size *c* bits
  - The VA data must be loaded in chunks of *c* bits.
  - Thus at any stage only *c* bytes of frame data can be modified
- Partition the memory frames into blocks such that there are *c* frames per block
- Read *c* top bytes from a block into FDR, modify them and write them back
- Involves *c* horizontal buses instead of buses for all bytes in the frame
- Fix *c*=8
  - Virtex, Virtex-II and Virtex-IV all have 8-bit wide configuration ports
  - Pin limitations will not allow port width to increase substantially

The New Architecture

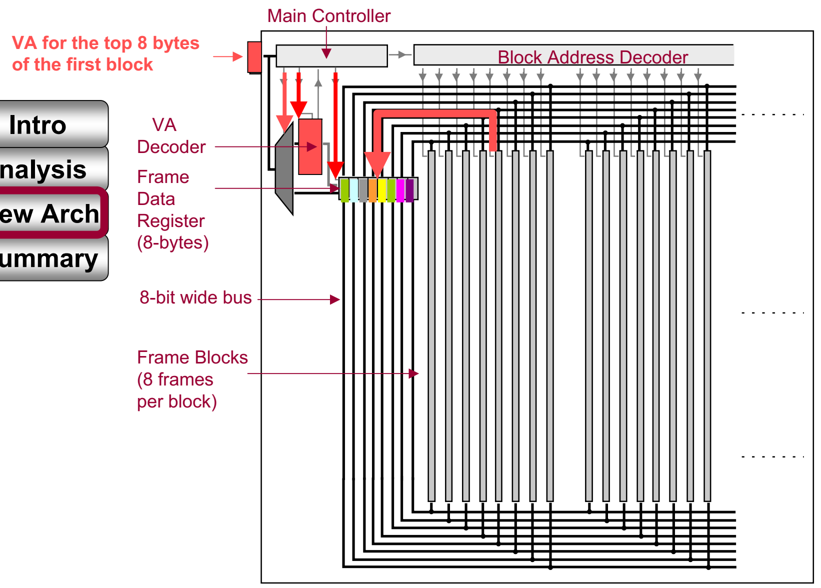
Intro
Analysis
New Arch
Summary


Main Controller
Configuration Port (8-bits)
Block Address Decoder
VA Decoder
Frame Data Register (8-bytes)
8-bit wide bus
Frame Blocks (8 frames per block)

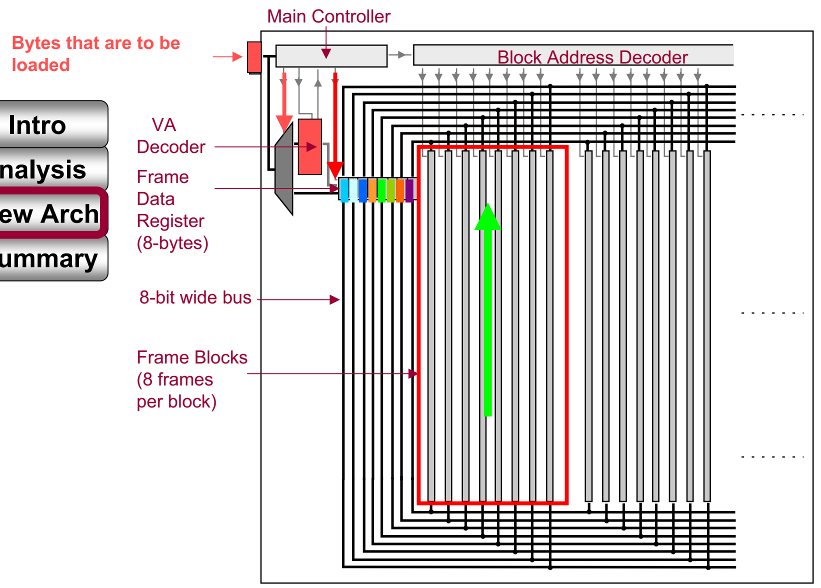
The Operation of the Memory


Main Controller
Starting Block Address + #consecutive blocks
Block Address Decoder
VA Decoder
Frame Data Register (8-bytes)
8-bit wide bus
Frame Blocks (8 frames per block)

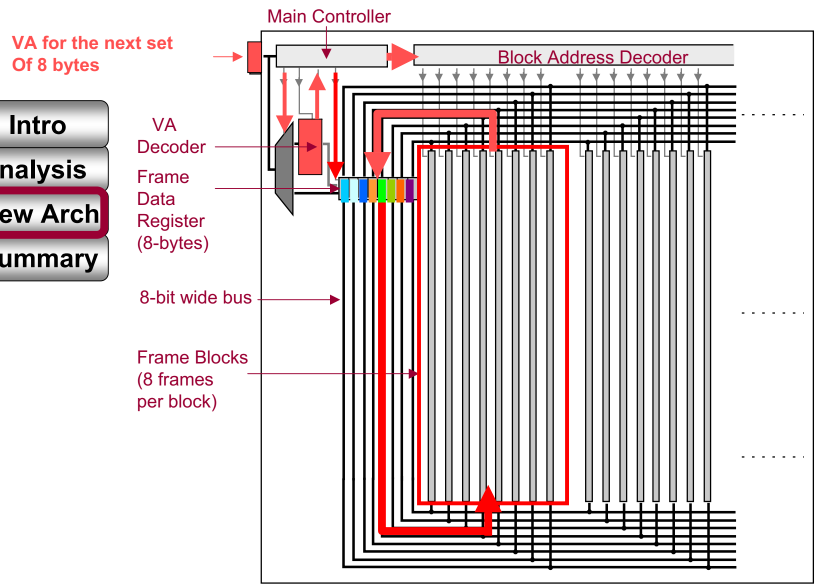
The Operation of the Memory


Main Controller
VA for the top 8 bytes of the first block
Block Address Decoder
VA Decoder
Frame Data Register (8-bytes)
8-bit wide bus
Frame Blocks (8 frames per block)

The Operation of the Memory

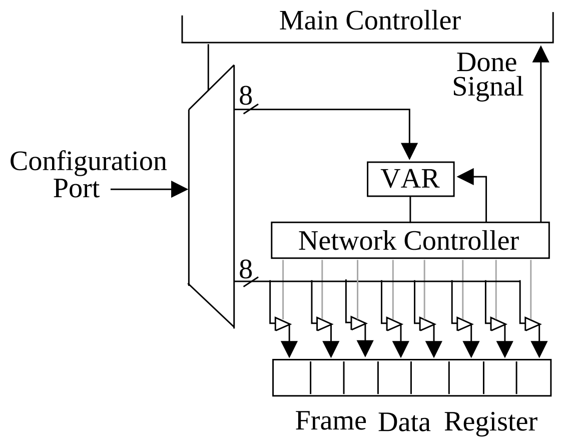

Main Controller
Bytes that are to be loaded
Block Address Decoder
VA Decoder
Frame Data Register (8-bytes)
8-bit wide bus
Frame Blocks (8 frames per block)

The Operation of the Memory


Main Controller
VA for the next set Of 8 bytes
Block Address Decoder
VA Decoder
Frame Data Register (8-bytes)
8-bit wide bus
Frame Blocks (8 frames per block)

The Vector Address Decoder

Intro
Analysis
New Arch
Summary



The Network Controller

Intro
Analysis
New Arch
Summary

- Let V be the 8 bits of the input vector address stored in the VAR. The goal is to generate *i* vectors such that $V = V_1$ ***xor*** V_2 …. ***xor*** V_i where *i* is the number of set bits in that portion of VA.
- Define a *mask register* (MA) such that

  MR[7] = VAR[7] + VAR[6]….VAR[0]

  $MR[j] = \overline{VAR[j+1]}.MAR[j+1],\ 6 \geq j \geq 0$
- The address signals are generated by successive XOR operation

  v_j = MR[j] ***xor*** MR[j+1], v0 = MR[0] ***xor*** VAR[0]
- The processed set bit in the VAR is cleared and the above cycle repeats
- A maximum of 8 gate delays that can be accommodated in a single cycle
- The done signal is generated as

  done = $\overline{VAR[7] + VAR[6]....VAR[0]}$ (3 gate delays)

Evaluating the New Design

Intro
Analysis
New Arch
Summary

- Additional VA will be needed if the user configuration does not span blocks of eight.
- For the set of benchmark circuits it was calculated that the DMA-VA provides about 62% reduction in the overall amount of configuration data.
  - The VA overhead decreases compared to the VA model because we have removed the VA corresponding to frames that are not loaded in the Virtex model
- Thus DMA-VA offers similar levels of configuration data reduction as the device-level VA.

Implementation Results

Intro
Analysis
New Arch
Summary

- The implementation details of Virtex are not known to us
  - 0.22μm, 5 metal layers, XCV100 is packaged in 27mm^2
- The current Virtex model and the new design were implemented in VHDL and Synopsis Design Compiler (v 2004.06) was used to synthesize it to a 90nm cell library
  - Target device was XCV100 (20 x 30 CLBs, 56bytes/per frame,1610 frames)
  - Max fan-out 32, V= 3.3volts
- Area
  - Difficulty in synthesizing the entire design
  - Synthesized main controller + decoders +8frames
    - The frame area was found to be almost linear in the number of frames
    - Each frame approximately adds 20,700μm^2
  - Current Virtex Results
    - Main controller = 70,377μm^2, FAD+ 8 frames = 156,742μm^2
    - Estimated total device (main controller excluded) = 3.32 x 10^7 μm^2 (or 33mm^2)
  - New Virtex Results
    - Main Controller = 2,592 μm^2, VAD = 3,458 μm^2, BAD+8frames = 319,630μm^2
    - Estimated total device (main controller excluded) = 3.34 x 10^{7}μm^2
    - Approximately 0.5% area increase compare to the base memory model
  - Note: As we do not have SRAM libraries, the area estimates are based on FF area. While absolute values might be bigger our design requires modest additional hardware relative to the base memory model

Implementation Results

- The Delay results suggest that the new design can be clocked at 50MHz with the main controller taking the longest time (20ns). The VAD delay is only 8ns. The current Virtex model is externally clocked at 33MHz
- As we have assumed that we can read/write to the destination frame registers in a single cycle the wire delays also need to be accounted for
  - As we could not synthesize the entire device we estimated the wire delays using Elmore delay formula. The values for the wire resistance and capacitance were found from the TSMC data sheets
  - It was estimated that up to 28,86 frames could be spanned in 20ns. Scalability issue will be discussed later
- Power
  - Using DC the power estimated for the basic design with 8 frames was 353mW (including cell internal, net switching and cell leakage)
  - The new design with 8 frames had a power consumption of 871mW.
  - Thus power increases by 59%.
  - However, the actual situation is more complicated
    - A recent study (Lorenz *et. al.* [FPL04]) has shown that energy wasted during FPGA reconfiguration is dominated by short-circuit and static power of the cells that are being reconfigured. The longer it takes to reconfigure the more energy is consumed even if the same amount of data is written to the configuration memory (more than a linear increase).
    - Thus faster reconfiguration is desirable from power perspective
    - This issue is currently being investigated

Scalability

- As the device grows in size the wire delays will become significant and single cycle read will be an unrealistic assumption.
- Solution
  - Partition the memory into configuration pages
    - Virtex-IV seems to already have implemented configuration page strategy
  - Address the configuration pages in a RAM style fashion
  - Replicate the DMA-VA memory in each of the configuration pages
    - The area needed by the controller and the decoders is fairly small compared to the memory array
    - Pipeline the configuration distribution

Address Compression

- The VA data for typical circuits contain many zeros
  - Can compress to further reduce the amount of data to be loaded
- Evaluated a well-known hierarchical compression scheme
  - 66% reduction in the amount of configuration data
- The corresponding HW decompressor contributed significant control delays
- Schemes for distributed decompression were considered but they turned out to be too complicated to be implemented in hardware

Related Work

- Several people have worked on reducing reconfiguration delay
  - Architectural research
    - Time multiplexed FPGA (Trimberger[97]). Involves doubling the configuration memory requirements
    - Pipeline reconfiguration (Schmit[97]). Local memory interconnect for pipelined FPGAs
  - Algorithmic research
    - Scheduling reconfigurations (Sarrafzadeh[03])
  - Configuration compression
    - Dictionary based compression up to 41%(Dandalis *et. al.* [01]). Requires significant on-chip memory for decompression
    - LZ77 based compression (Li *et al.* [01]). Reduction up to 75%. Assumes a RAM style configuration distribution network.
    - LZ based compression [*Ju et al.* [04]]. Compression up to 76%. No H/W decompressor described.
  - Configuration caching
    - Mainly in the context of tightly coupled gate arrays (e.g. Li *et.al.* [00] and Sadhir *et al.* [01])

Conclusions and Future Work

- A new configuration memory architecture has been developed that reduces the reconfiguration time of an FPGA by 2.5X for a set of benchmark circuits at modest additional hardware cost
- Techniques for incorporating published compression methods into our methodology
  - We applied Huffman compression on the benchmark partial configurations (frame data + VA data) and found up to 87% reduction in the amount of data (LZ77 gave a 78% reduction)
    - A corresponding reduction in decompression in not possible unless bandwidth mismatch problem is solved
  - Study the feasibility of distributing the decompressors to maintain a constant throughput at the configuration port
  - Study the feasibility of inter-frame configuration re-use