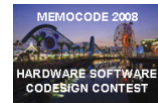




M4- Crypto -Sorter

VJ Sananda
AMD

Outline



2

- Team/Goals
- Reference Design Investigation
- Optimized Algorithm Development
- HW/SW Partitioning
- Define HW Architecture
- Theory of Operation
- Performance Analysis
- Design Implementation
- Results
- Improvements

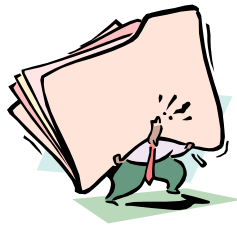
Insert Paper Title here

Team / Goals

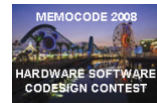


3

- Had expected a team of 5
- All dropped out because of work related commitments
- Army - Of – One
- Must Work .. The greatest architecture is worthless if it doesn't.



FPGA Platform



4

- XUPV2Pro Board
- 300 MHz PPC
- Cache Enabled
- PLB Bus Clock 100 MHz
- 4KB IOCM
- Stdin/Stdout via OPB_Uartlite (no license restrictions)
- DDR 512Mb Dual-Rank , PLB
- Developed in ISE/EDK 7.1i, Ported to 91i final week

Insert Paper Title here

Reference Design Algorithm Investigation



5

- Ran gprof using general purpose CPU
- 78% of time spent in AES crypto calls, because of in place sort.
- Every record access and swap uses AES keygen function
- Modified RefDesign to first decrypt, then sort, then encrypt. Performance increased 7X.
- Follow similar strategy for HW accelerator

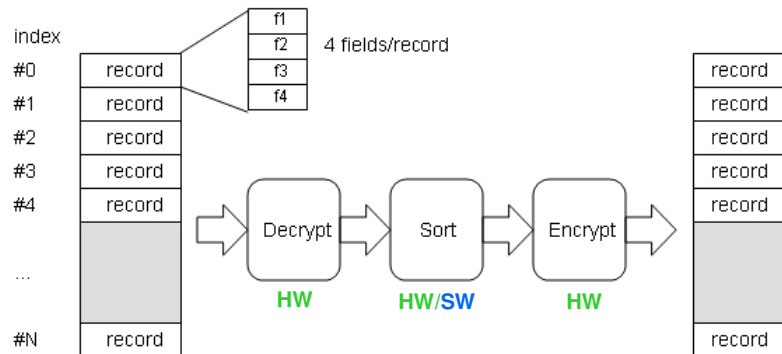
Optimized Algorithm Development



6

- Accelerate encryption/decryption in HW, however what can we do about sorting ?
- Max Datasize of 262144 records = 4Mbytes > Localized Block RAM (306 Kbytes)
- Overhead of repeated access to DRAM could add up, so decided against 100% HW based sort.
- Partition sort between HW and SW

HW/SW Partitioning



Optimized Algorithm Development

- Partitioning sort between HW/SW
- Bottom up Merge Sort is a good fit.
- HW will decrypt and sort a block records
- SW on PPC will then do a bottom up merge sort using these sorted blocks
- Final Encryption Pass using the HW to produce results.

Optimized Algorithm Development

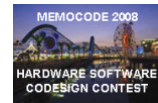


9

- Developed SW version of Merge Sort,
- Non-recursive (loop unrolled).
- Function call defined in `mssort.c`
 - `int merge_all_sorted_blocks(int blocksize,)`
- Number of sorted records in each block = blocksize
- Easy to incorporate improved HW sort capabilities, either
 - Larger blocksizes
 - Multiple HW sorters running in parallel



Define HW Architecture



10

- Decided on a 64 bit PLB peripheral with FIFOs for Read and Write
- Allows scaling up to multiple accelerators. Note that optimized algorithm allows this.
- Start with 64 bit PLB slave peripheral.
- Incorporate bus-mastering as time permits
- FIFO Depth = 128 (64bit words) which is equivalent to 64 Records
- Decided 64 would be the upper limit on HW sort capabilities. (Currently at 4)
- 4 Registers for Control and Status

Insert Paper Title here

Define HW Architecture



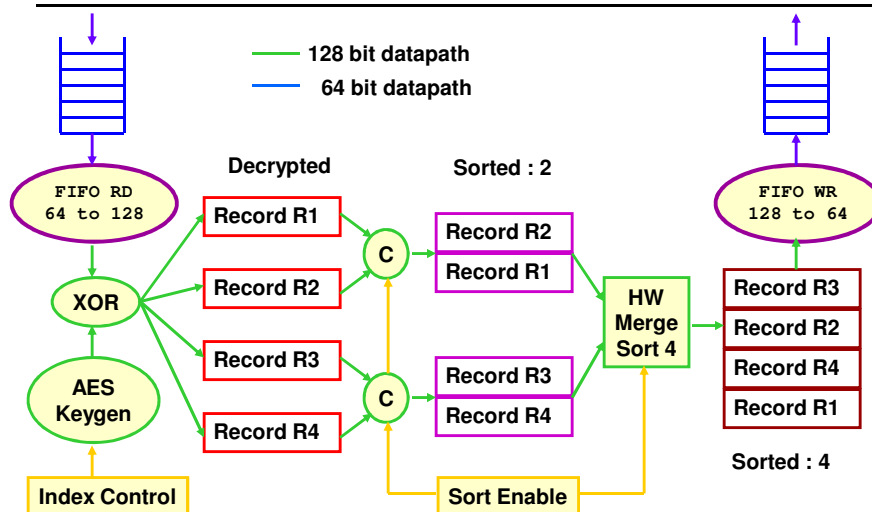
11

- Each Record consists of 4 , 32 bit fields.
- Record Compare equivalent to 128 bit compare with f1 field towards the MSB and f4 towards LSB
- That is $R_128 = \{ R.f1, R.f2, R.f3, R.f4 \}$
- Record R1 < Record R2 if $R1_128 < R2_128$
- So HW uses 128 bit data path
- For PLB peripheral FIFO writes and reads $\{R.f1, R.f2\}$ is the 1st , followed by $\{R.f3, R.f4\}$

HW Architecture: Block Diagram



12

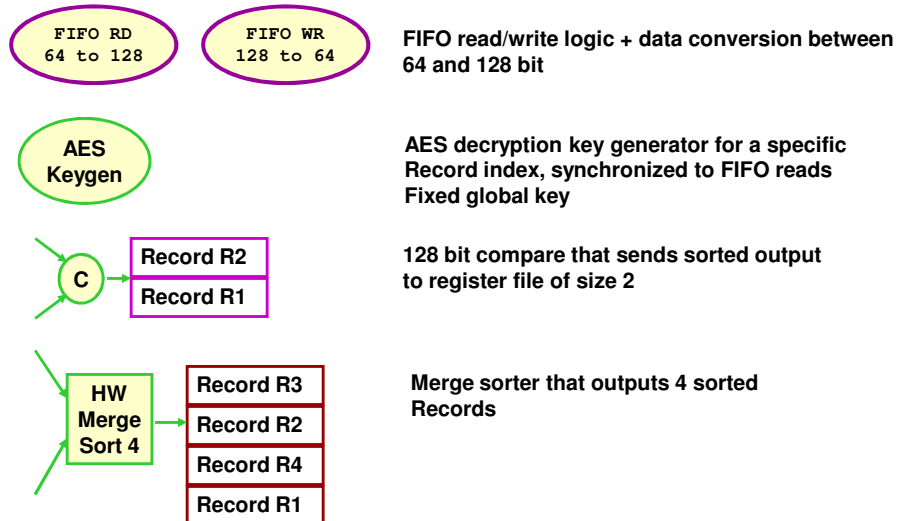


Insert Paper Title here

HW Architecture: Sub-blocks



13



HW Merge Sort 4: Pseudo Code



14

- Definitions
 - **Reg_File**: register file of size 2 holding sorted Records. That is $\text{Reg_File}[0] < \text{Reg_File}[1]$
 - **Reg_File_Ptr** is a pointer referencing **Reg_File**
- In HW we build 2 **Reg_File** structures each holding 2 records , sorted using a simple 128 bit compare
 - **Reg1_File** and **Reg2_File**
 - Associated pointers are **Reg1_File_Ptr** and **Reg2_File_Ptr**
 - Valid values for **Reg*_File_Ptrs** are 0 and 1, use an extra bit ($\text{Reg_File_Ptr} == 2$) to detect when we have output both values

HW Merge Sort 4: Pseudo Code



15

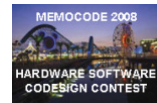
```
while (Entries still to be output from Reg*_Files ) {

    if ( Reg2_File not empty AND Reg1_File empty) {
        Output Reg2_File[Reg2_File_Ptr++];
        continue;
    }

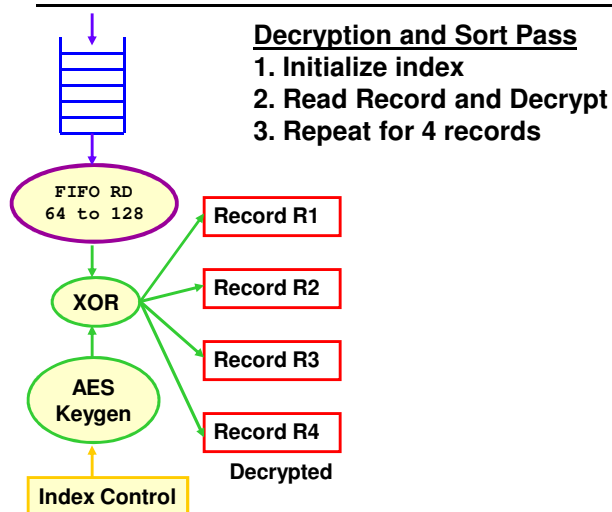
    if ( Reg1_File not empty AND Reg2_File empty) {
        Output Reg1_File[Reg1_File_Ptr++];
        continue;
    }

    if (Reg1_File[Reg1_File_Ptr] < Reg2_File[Reg2_File_Ptr])
        Output Reg1_File[Reg1_File_Ptr++];
    else
        Output Reg2_File[Reg2_File_Ptr++];
}
```

Theory of Operation



16

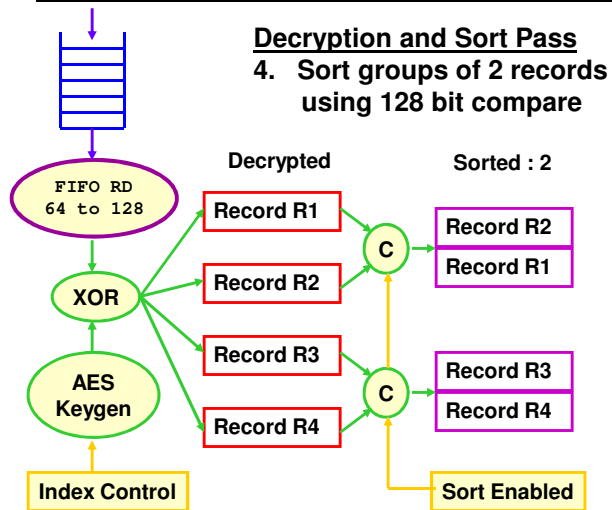


Insert Paper Title here

Theory of Operation



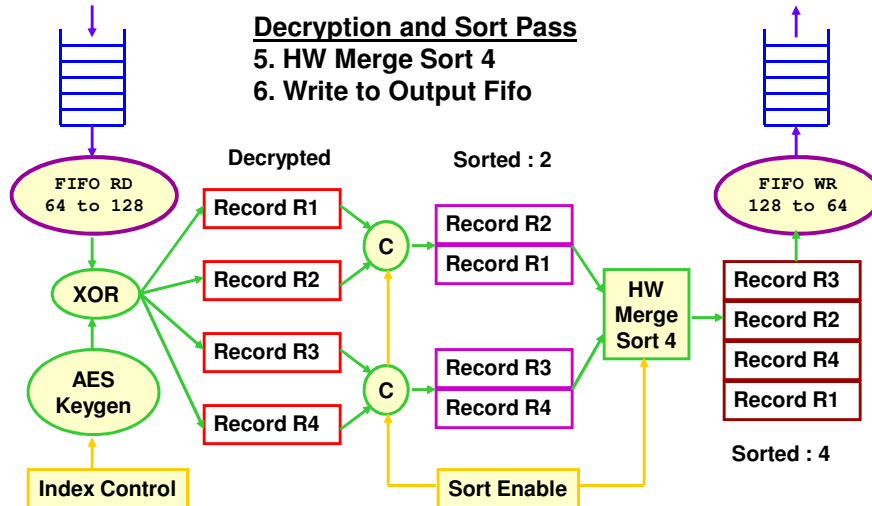
17



Theory of Operation



18

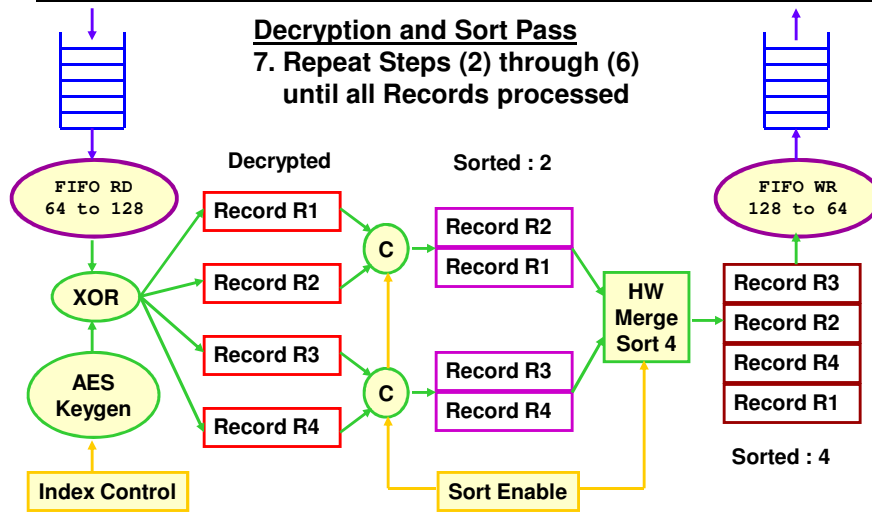


Insert Paper Title here

Theory of Operation



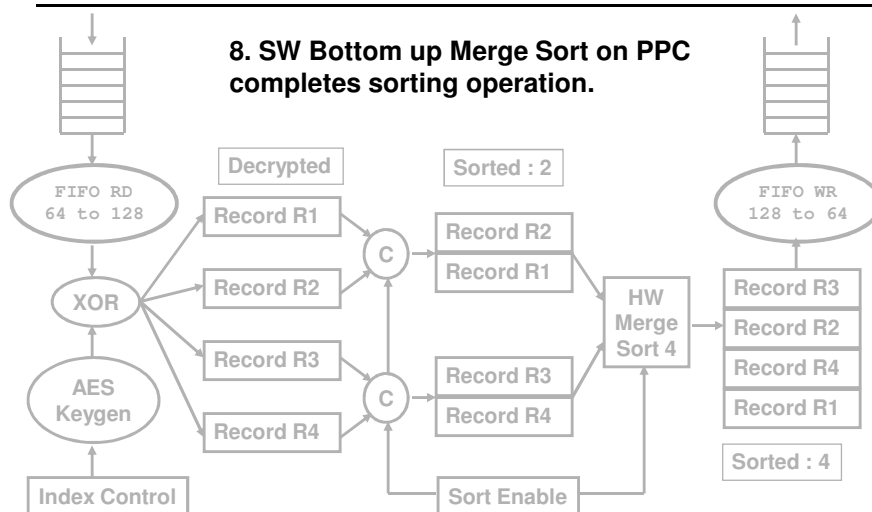
19



Theory of Operation



20

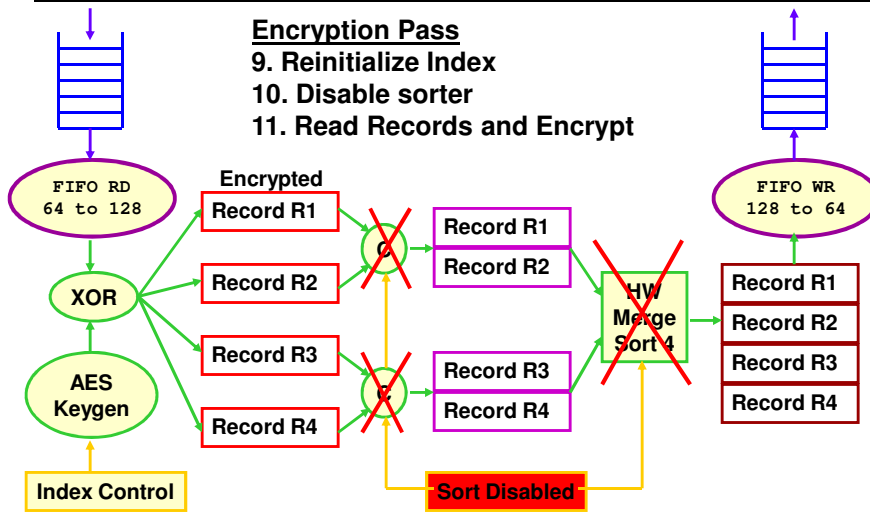


Insert Paper Title here

Theory of Operation



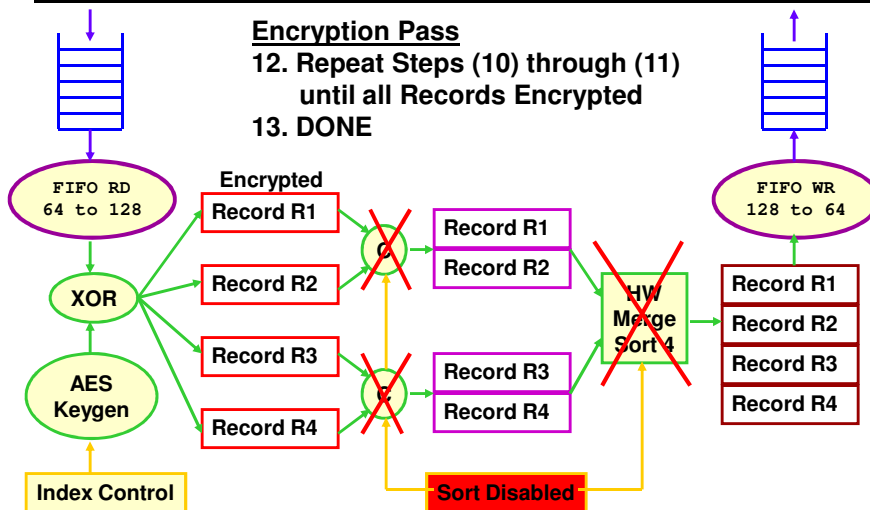
21



Theory of Operation



22



Insert Paper Title here



Performance Analysis

- PLB FIFO Drain & Fill Rate by accel_sort.v
- FIFO Read and Data Conversion : 2 PLB Cycles
- Keygen = 10 PLB Cycles
- For 4 Records, Read and Decrypt : $(10+2) * 4 = 48$ PLB Cycles
- 2 Record compare = 2 PLB Cycles
- 2 cycles per Record for Merge Sort and Output, = $2 * 4 = 8$ PLB Cycles
- Total PLB Cycles = $48 + 2 + 8 = 58$ Cycles.
- Total in **CPU Cycles** = $58 * 3 = 174$



Performance Analysis

- PLB FIFO Fill Rate from DRAM. Initiated by PPC : 5 PLB Cycles + 2 CPU Cycles + DRAM access
- PLB FIFO Drain Rate from DRAM. 6 PLB Cycles + 3 CPU Cycles + DRAM access
- For 4 operands, in CPU Cycles $((5*3+2)+(6*3+2)) * 4 = 148$ cycles + 2 * DRAM access
- If we assume a 50 ns DRAM access and arbitration time
- Total in **CPU Cycles** = $148 + (5*3)*2 = 178$

Performance Analysis



25

- The PLB FIFO fill and drain rate by the PPC is close to (but smaller) than that by the HW accelerator.
- So should be able to Read Results from the FIFO, without stalling.
- Also indicates that to service multiple HW accelerators will require a much faster DRAM to PLB FIFO data path
- For now, aim is for this single HW accelerator to provide the “best - bang - for - the - buck”.

Design Implementation



26

- For AES Keygen used AES Core from Open Cores.org (author Rudolf Usselman)
- Is free-running, Added logic to set start index, reset index.
- Qualified AES Core using Keys in reference design
- Quick XST Synthesis run to confirm 100MHz+ speed
- AES Core performs 1 encryption in 10 cycles.



Design Implementation

- Used Create/Import IP Wizard in EDK to generate Core-Connect - PLB interface files in VHDL.
- HW Accelerator (accel_sort.v) developed in Verilog, instantiated in user_logic.v
- Due to Manpower limitations, had to aim for rtl code being Correct by Construction
- Used robust handshake protocols , ready – push/pull
- No “just in time” logic, which can be very brittle.



Design Implementation

- **accel_sort.v** has the master control machine.
- Initiates FIFO read using **fifo_rd_64_to_128.v**
- Pulls key from **aes_keygen.v** for decrypt/encrypt
- Does 2 Record compare
- Does 4 Record Merge Sort
- Initiates FIFO write using **fifo_wr_128_to_64.v**
- 1866 non blank lines of rtl verilog code
 - 762 in AES rtl IP , 1104 lines of new rtl
- 298 non blank lines of testbench verilog code

Design Imp: Changes made to Ref Design Files



29

- Primary change made to main_tc.v is to replace call to sortrecord()
- In EDK 9.1i xbasic_types.h in PPC405_0/include has conflicting typedefs for u32 and u8 (also defined in aes_core.h)
- Changed aes_core.h
 - u32 → _u32_
 - u8 → _u8_
- 477 non blank lines of driver code

Changes made to Ref Design Files : main_tc.c



30

```
//VJS:Added hw merge sort
#include "sortrecord_hw_plb.h"
#include "msort.h"

//VJS:Changed cache enable from 0xf0000000 to
0xc0000000
XCache_EnableICache(0xc0000000);

//VJS:Comment out refcode and add hw accel call
//sortrecord_tc(6+4*i);
sortrecord_hw_plb(6+4*i);
```

Essentially only replaced sortrecord_tc call.



Results: Running/Debug

- Bitstream configured using Bootloop
- Configuration Bitstream : **download.bit**
 - Click on EDK download icon
- Use XMD to download and run executable:
 - dow metric/executable.elf
 - run
- SW Project with test code is **metric**
- Submission is for ISE/EDK 9.1i with latest service packs for ISE and EDK



Results: RefDesign Vs. M4-CryptoSort



Results : Resource Utilization

- Number of Slice Flip Flops: **15%**
– 4,357 out of 27,392
- Number of occupied Slices: **32%**
– 4,482 out of 13,696
- Total Number of 4 input LUTs: **24%**
– 6,784 out of 27,392
- Number of Block RAMs: **10%**
– 14 out of 136



Results: Sample Output

```



Sorting starts
Case (rand, pwr= 6) : Elapsed      220,  correct
Case ( rot, pwr= 6) : Elapsed      216,  correct
Case (rand, pwr= 10) : Elapsed     5902,  correct
Case ( rot, pwr= 10) : Elapsed     5900,  correct
Case (rand, pwr= 14) : Elapsed    119071,  correct
Case ( rot, pwr= 14) : Elapsed    118352,  correct
Case (rand, pwr= 18) : Elapsed   2285336,  correct
Case ( rot, pwr= 18) : Elapsed   2265987,  correct
Relative Geometric Mean: 33.003149
Sorting completed
  
```



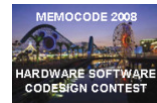
Results



35

- Works !
- Passes metrics test bench
- Score ~33 
-  Ref Design Score ~1
- In case you were wondering:
M4 Crypto Sort = **M**ergeSort - Block**4** Crypto Sort

Improvements



36

- Scale using multiple HW accelerators, will require a bus mastering solution (due to time and manpower constraints had to stop with a PLB slave implementation).
- Explore sorting larger blocks in HW using say a bitonic sort.
- With a fast DRAM access path, can pre-compute the encryption/decryption keys once, and then fetch as needed.