JPEG Compression

Accelerating Systems with Programmable Logic Components

By Oscar Castro - Henrik Schulze

JPEG Compression: Task

Calculate the number of consecutive occurrences with a determined sequence on the matrix.

JPEG Compression: Task

Calculate the number of consecutive occurrences with a determined sequence on the matrix.

Matrix 8x8

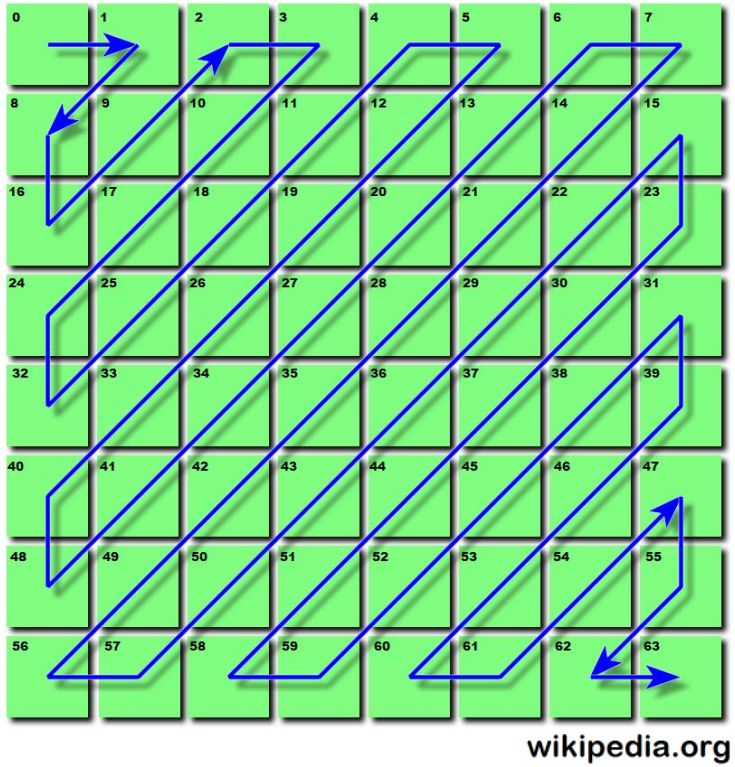
$$A = \begin{bmatrix} 5 & 5 & 3 & 3 & 5 & 5 & 7 & 7 \\ 5 & 3 & 3 & 3 & 5 & 7 & 0 & 0 \\ 5 & 3 & 3 & 4 & 7 & 4 & 0 & 0 \\ 5 & 5 & 4 & 7 & 0 & 0 & 0 & 0 \\ 5 & 6 & 7 & 8 & 0 & 0 & 0 & 0 \\ 6 & 6 & 8 & 0 & 0 & 0 & 0 & 0 \\ 6 & 8 & 0 & 0 & 0 & 0 & 0 & 0 \\ 8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

JPEG Compression: Task

Calculate the number of consecutive occurrences with a determined sequence on the matrix.

Matrix
8x8

$$A = \begin{bmatrix} 5 & 5 & 3 & 3 & 5 & 5 & 7 & 7 \\ 5 & 3 & 3 & 3 & 5 & 7 & 0 & 0 \\ 5 & 3 & 3 & 4 & 7 & 4 & 0 & 0 \\ 5 & 5 & 4 & 7 & 0 & 0 & 0 & 0 \\ 5 & 6 & 7 & 8 & 0 & 0 & 0 & 0 \\ 6 & 6 & 8 & 0 & 0 & 0 & 0 & 0 \\ 6 & 8 & 0 & 0 & 0 & 0 & 0 & 0 \\ 8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$



sequence

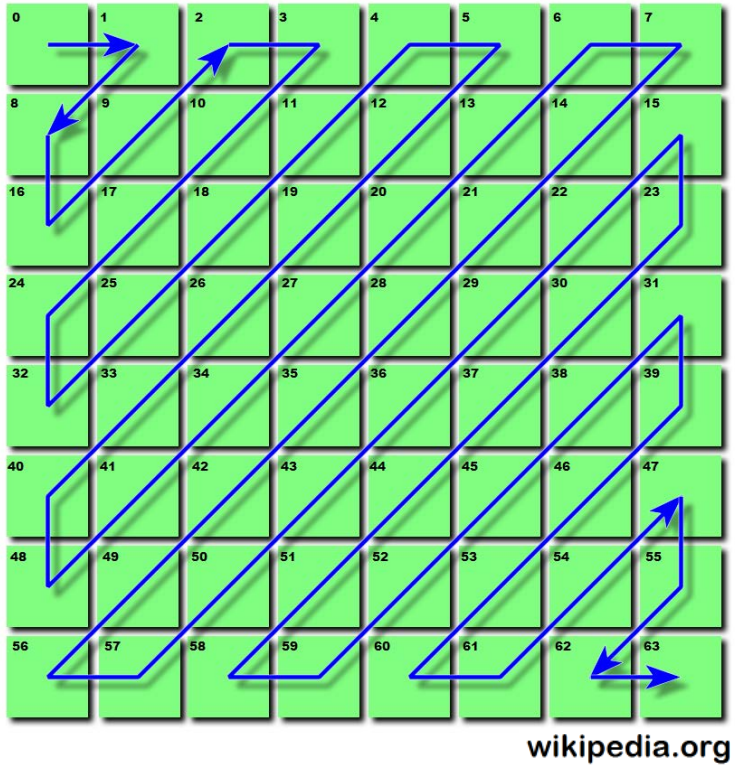
JPEG Compression: Task

Calculate the number of consecutive occurrences with a determined sequence on the matrix.

Matrix 8x8

$$A = \begin{bmatrix} 5 & 5 & 3 & 3 & 5 & 5 & 7 & 7 \\ 5 & 3 & 3 & 3 & 5 & 7 & 0 & 0 \\ 5 & 3 & 3 & 4 & 7 & 4 & 0 & 0 \\ 5 & 5 & 4 & 7 & 0 & 0 & 0 & 0 \\ 5 & 6 & 7 & 8 & 0 & 0 & 0 & 0 \\ 6 & 6 & 8 & 0 & 0 & 0 & 0 & 0 \\ 6 & 8 & 0 & 0 & 0 & 0 & 0 & 0 \\ 8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

"Zigzag" order



sequence

JPEG Compression: Task

Calculate the number of consecutive occurrences with a determined sequence on the matrix.

Send result

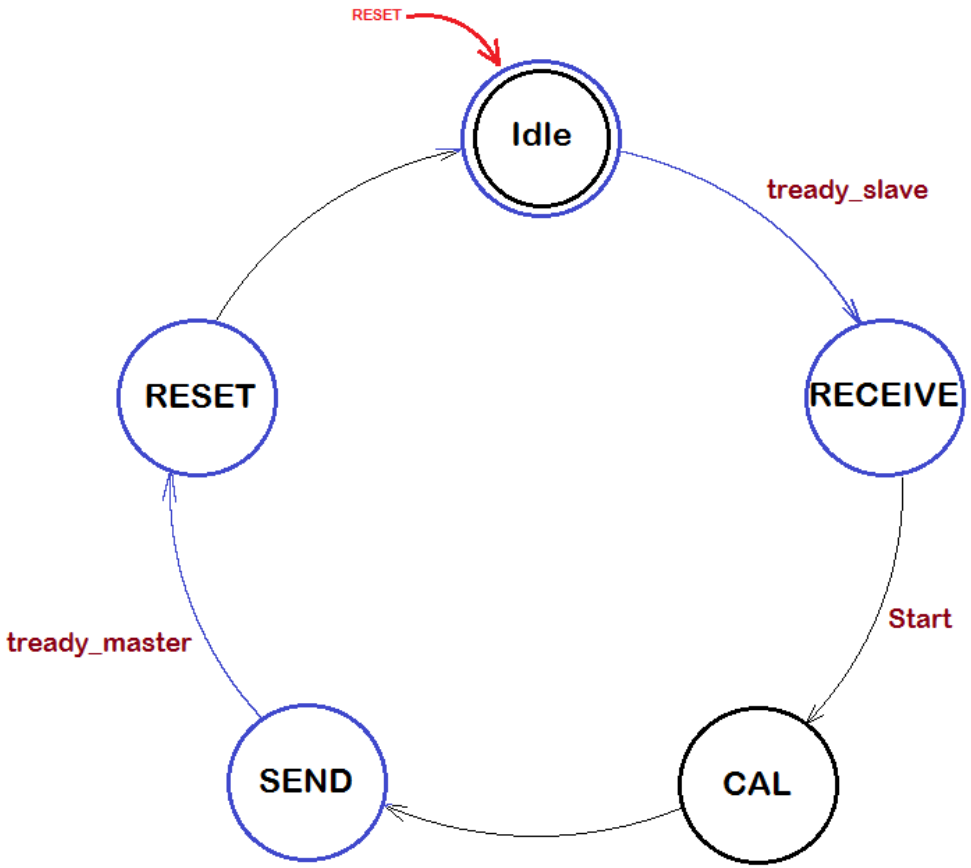
Matrix 8x8

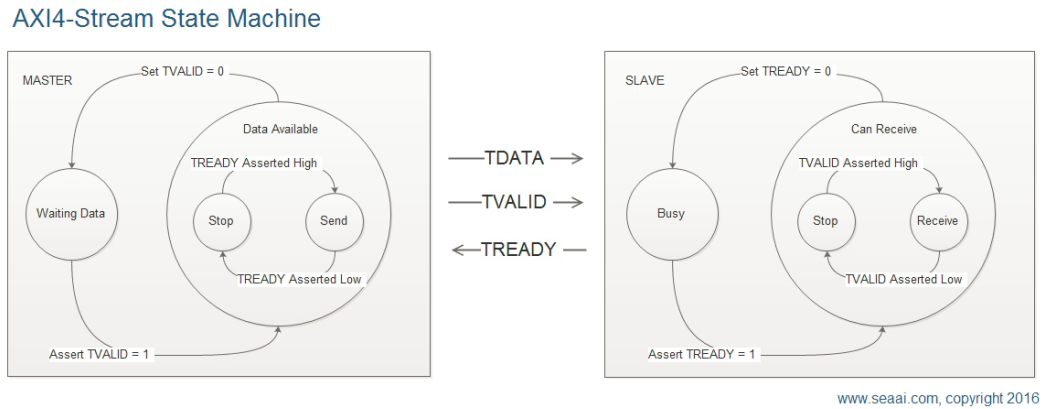
$$A = \begin{bmatrix} 5 & 5 & 3 & 3 & 5 & 5 & 7 & 7 \\ 5 & 3 & 3 & 3 & 5 & 7 & 0 & 0 \\ 5 & 3 & 3 & 4 & 7 & 4 & 0 & 0 \\ 5 & 5 & 4 & 7 & 0 & 0 & 0 & 0 \\ 5 & 6 & 7 & 8 & 0 & 0 & 0 & 0 \\ 6 & 6 & 8 & 0 & 0 & 0 & 0 & 0 \\ 6 & 8 & 0 & 0 & 0 & 0 & 0 & 0 \\ 8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

4,5 - 5,3 - 3,5 - 2,3 ⋯

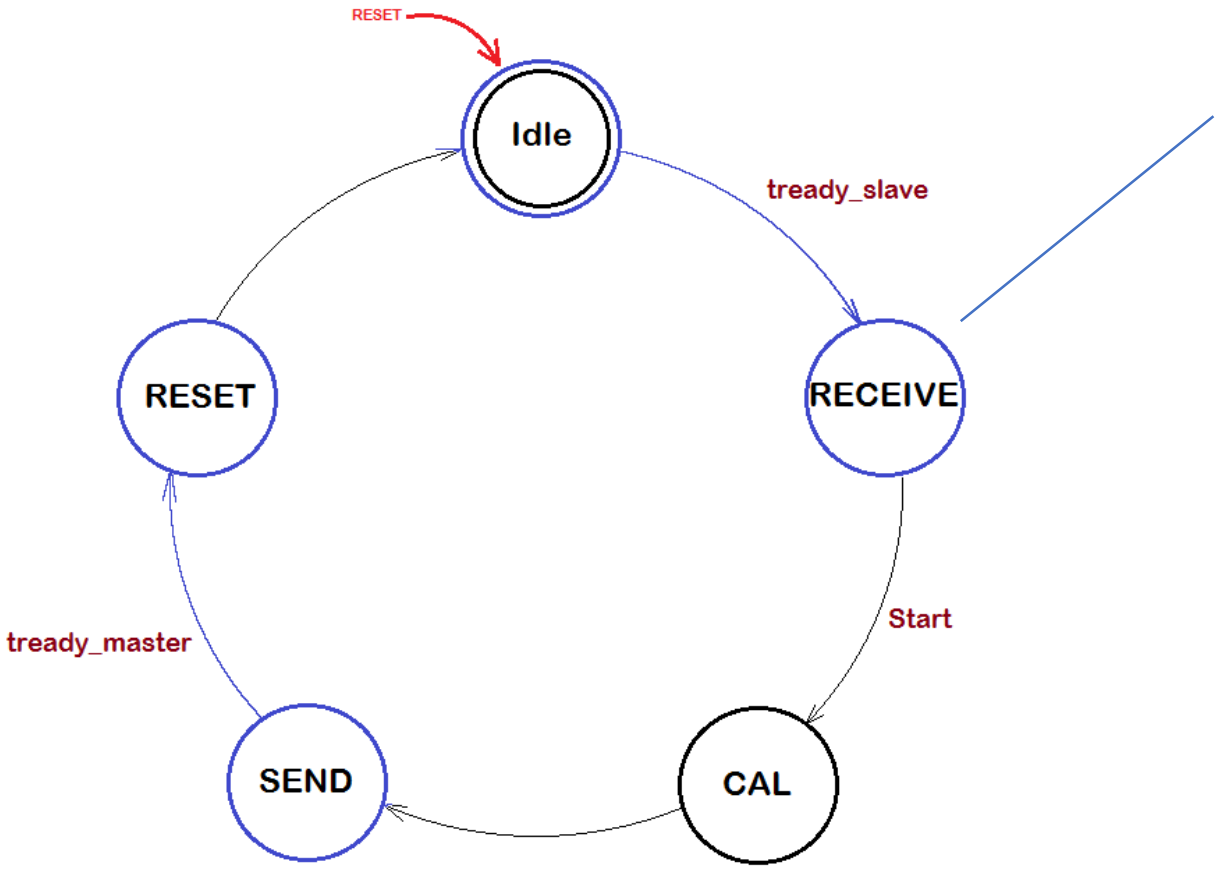
4	5	5	3	3	5	2	3	⋯

JPEG Compression: State Machine

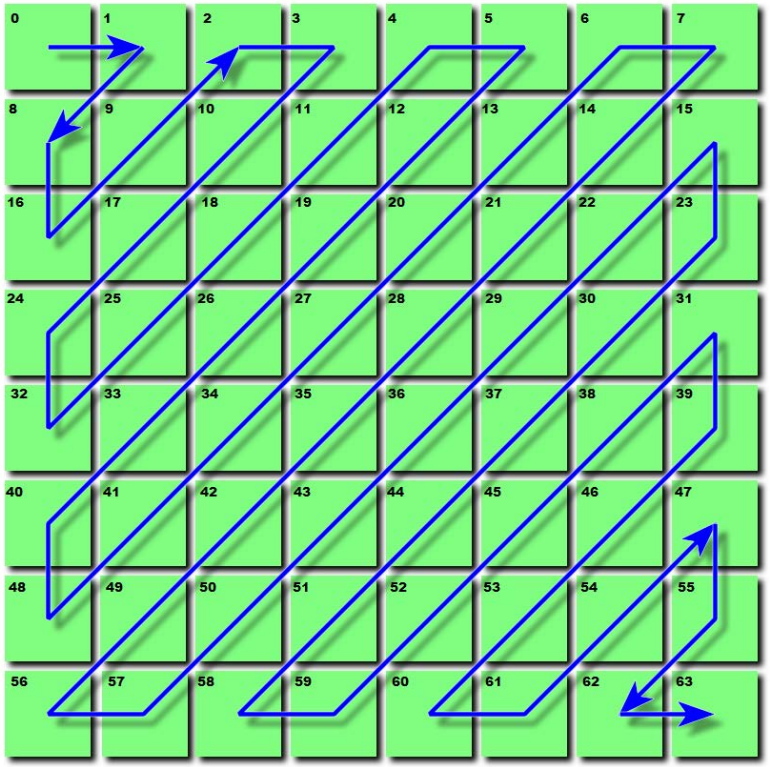




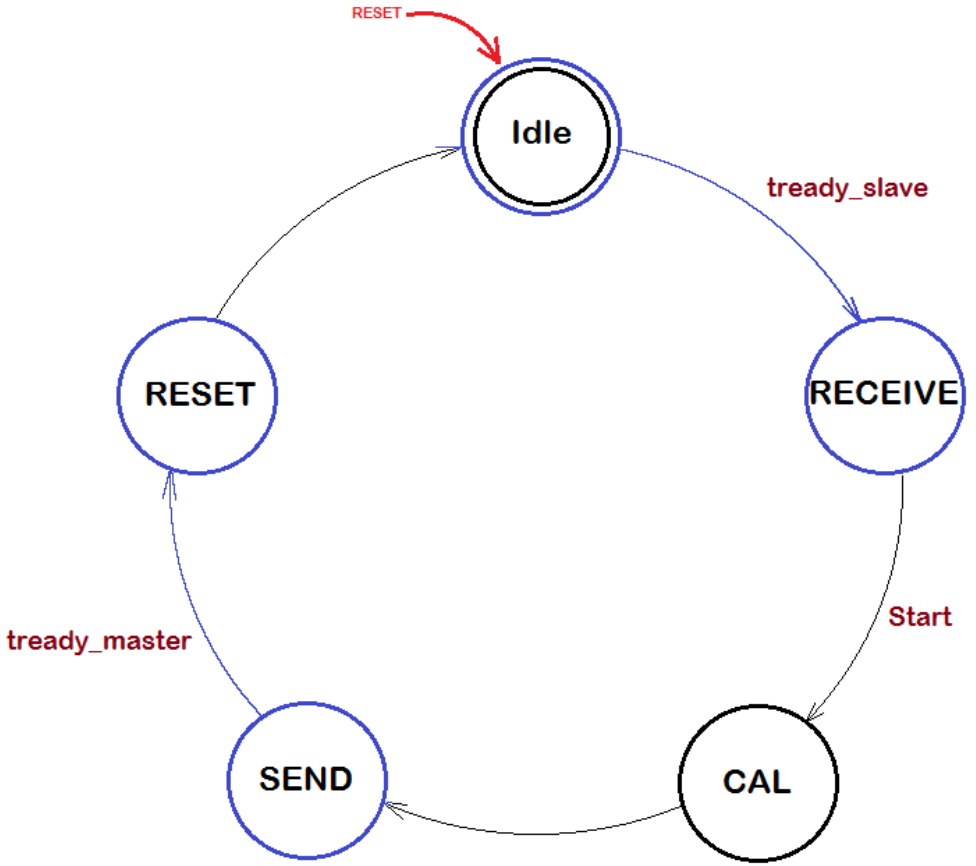
JPEG Compression: State Machine



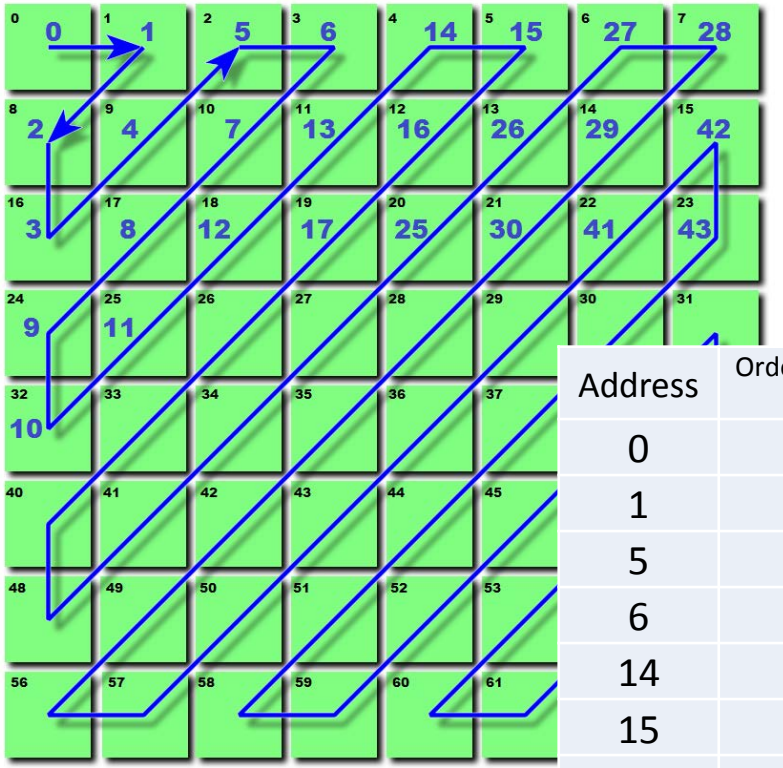
1. Receive the data and at the same time the data are ordained.



JPEG Compression: State Machine

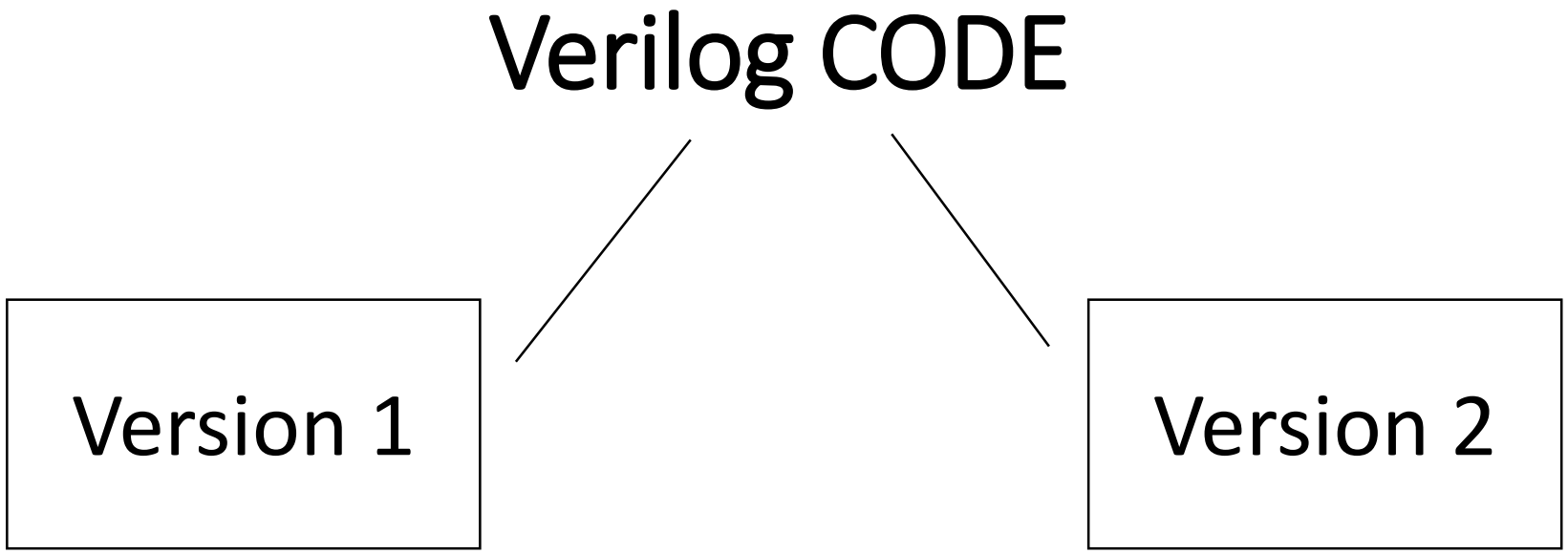


1. Receive the data and at the same time the data are ordained.



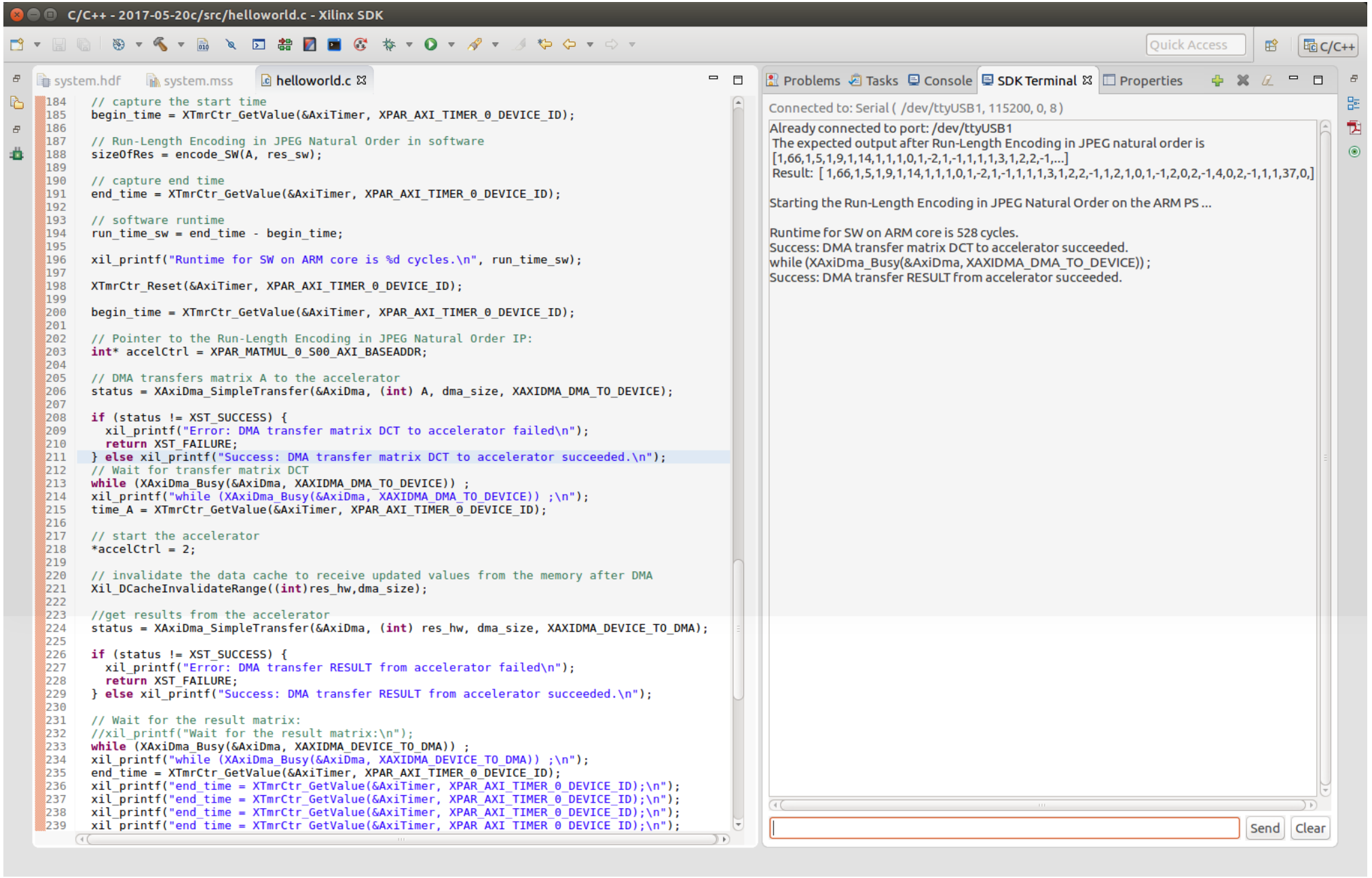
Address	Order arrived data
0	0
1	1
5	2
6	3
14	4
15	5
27	6
…	…

JPEG Compression: HLS (High level synthesis)



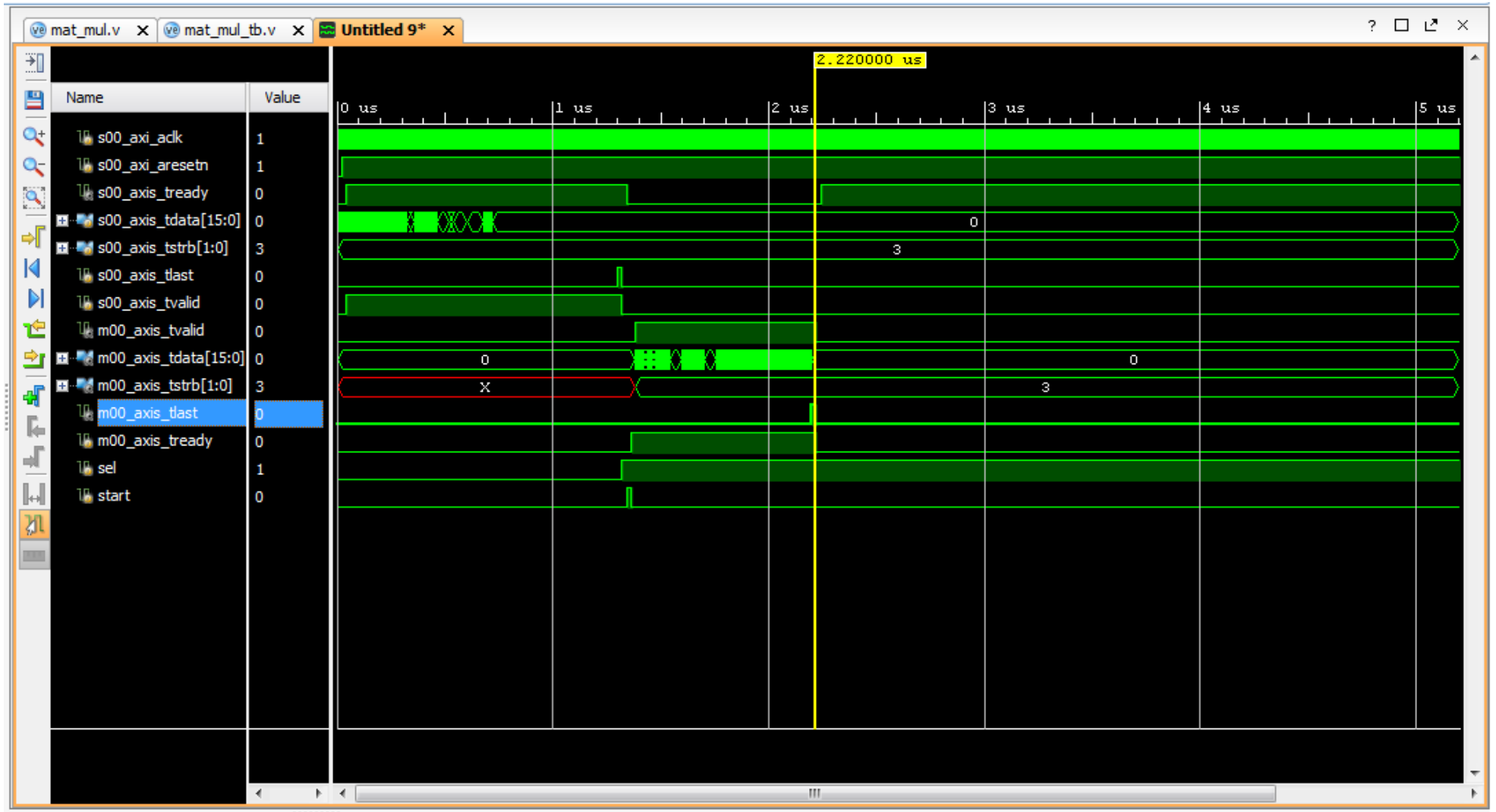
Let’s see the Verilog code….

JPEG Compression: C Code

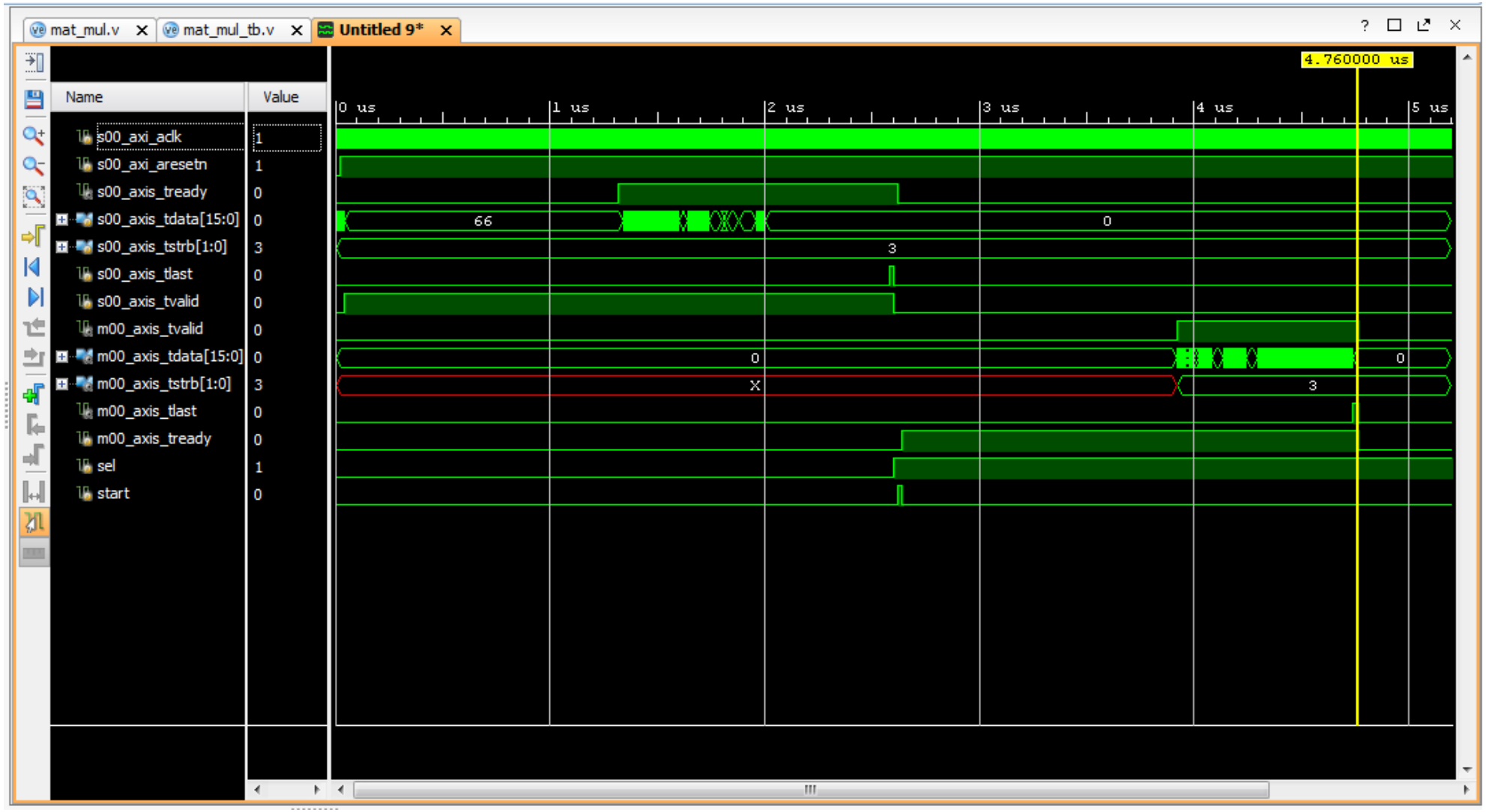



```
184   // capture the start time
185   begin_time = XTmrCtr_GetValue(&AxiTimer, XPAR_AXI_TIMER_0_DEVICE_ID);
186
187   // Run-Length Encoding in JPEG Natural Order in software
188   sizeOfRes = encode_SW(A, res_sw);
189
190   // capture end time
191   end_time = XTmrCtr_GetValue(&AxiTimer, XPAR_AXI_TIMER_0_DEVICE_ID);
192
193   // software runtime
194   run_time_sw = end_time - begin_time;
195
196   xil_printf("Runtime for SW on ARM core is %d cycles.\n", run_time_sw);
197
198   XTmrCtr_Reset(&AxiTimer, XPAR_AXI_TIMER_0_DEVICE_ID);
199
200   begin_time = XTmrCtr_GetValue(&AxiTimer, XPAR_AXI_TIMER_0_DEVICE_ID);
201
202   // Pointer to the Run-Length Encoding in JPEG Natural Order IP:
203   int* accelCtrl = XPAR_MATMUL_0_S00_AXI_BASEADDR;
204
205   // DMA transfers matrix A to the accelerator
206   status = XAxiDma_SimpleTransfer(&AxiDma, (int) A, dma_size, XAXIDMA_DMA_TO_DEVICE);
207
208   if (status != XST_SUCCESS) {
209     xil_printf("Error: DMA transfer matrix DCT to accelerator failed\n");
210     return XST_FAILURE;
211   } else xil_printf("Success: DMA transfer matrix DCT to accelerator succeeded.\n");
212   // Wait for transfer matrix DCT
213   while (XAxiDma_Busy(&AxiDma, XAXIDMA_DMA_TO_DEVICE)) ;
214   xil_printf("while (XAxiDma_Busy(&AxiDma, XAXIDMA_DMA_TO_DEVICE)) ;\n");
215   time_A = XTmrCtr_GetValue(&AxiTimer, XPAR_AXI_TIMER_0_DEVICE_ID);
216
217   // start the accelerator
218   *accelCtrl = 2;
219
220   // invalidate the data cache to receive updated values from the memory after DMA
221   Xil_DCacheInvalidateRange((int)res_hw,dma_size);
222
223   //get results from the accelerator
224   status = XAxiDma_SimpleTransfer(&AxiDma, (int) res_hw, dma_size, XAXIDMA_DEVICE_TO_DMA);
225
226   if (status != XST_SUCCESS) {
227     xil_printf("Error: DMA transfer RESULT from accelerator failed\n");
228     return XST_FAILURE;
229   } else xil_printf("Success: DMA transfer RESULT from accelerator succeeded.\n");
230
231   // Wait for the result matrix:
232   //xil_printf("Wait for the result matrix:\n");
233   while (XAxiDma_Busy(&AxiDma, XAXIDMA_DEVICE_TO_DMA)) ;
234   xil_printf("while (XAxiDma_Busy(&AxiDma, XAXIDMA_DEVICE_TO_DMA)) ;\n");
235   end_time = XTmrCtr_GetValue(&AxiTimer, XPAR_AXI_TIMER_0_DEVICE_ID);
236   xil_printf("end_time = XTmrCtr_GetValue(&AxiTimer, XPAR_AXI_TIMER_0_DEVICE_ID);\n");
237   xil_printf("end_time = XTmrCtr_GetValue(&AxiTimer, XPAR_AXI_TIMER_0_DEVICE_ID);\n");
238   xil_printf("end_time = XTmrCtr_GetValue(&AxiTimer, XPAR_AXI_TIMER_0_DEVICE_ID);\n");
239   xil_printf("end_time = XTmrCtr_GetValue(&AxiTimer, XPAR_AXI_TIMER_0_DEVICE_ID);\n");
```

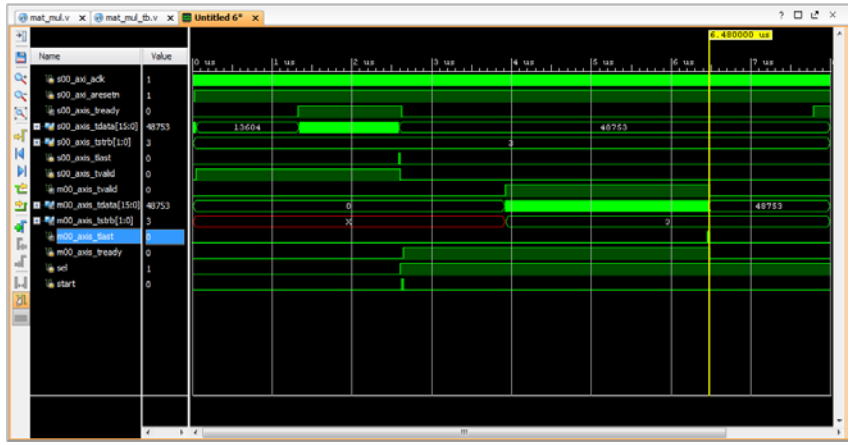
JPEG Compression: Results



JPEG Compression: (optimization)Results



JPEG Compression: Theoretical Results



Simulation time

Version 1

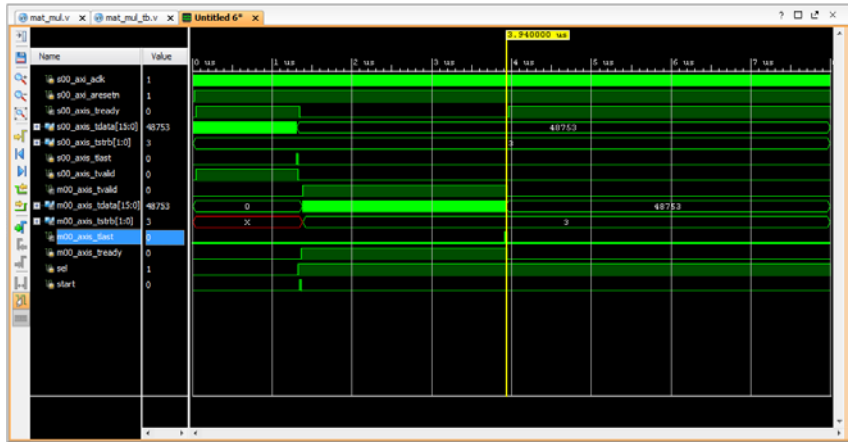
328 Cycles

Speedup

1.5

CPU Time

492 - 534 Cycles



Version 2

210 Cycles

Speedup

2.342

Problems …

L I M E

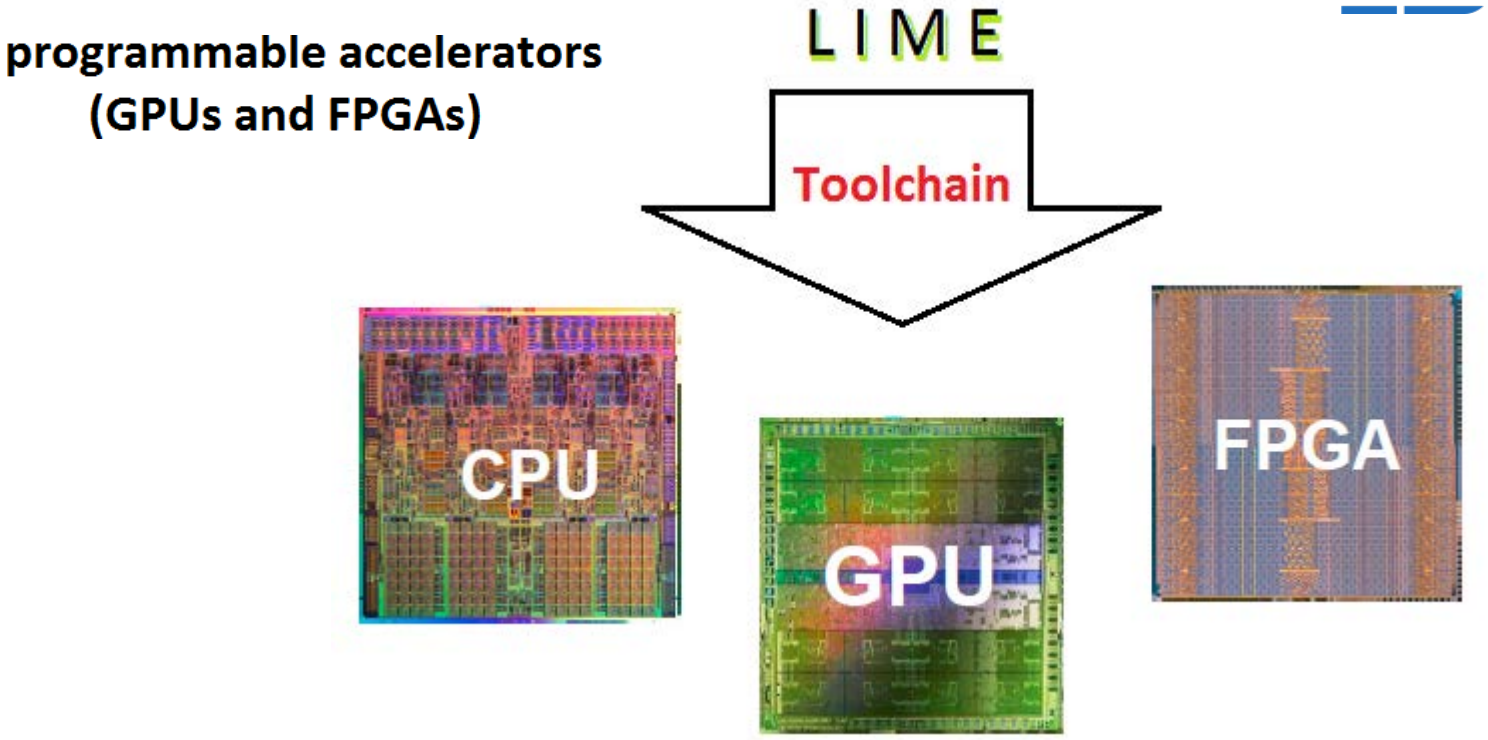
The Liquid Metal Programming Language

LIME: Introduction

Goal: The Liquid Metal project is a compiler and runtime system for heterogeneous architectures with a single and unified programming language.

LIME: Introduction

Goal: The Liquid Metal project is a compiler and runtime system for heterogeneous architectures with a single and unified programming language.

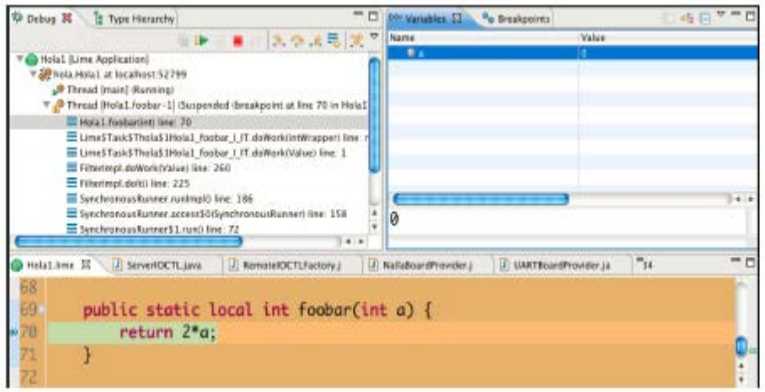


LIME: Introduction

Goal: The Liquid Metal project is a compiler and runtime system for heterogeneous architectures with a single and unified programming language.

IBM

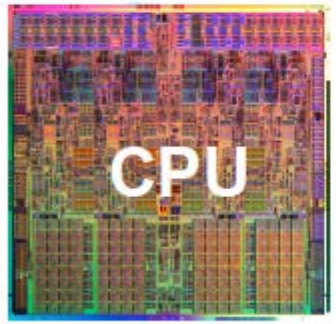
IDE, Compiler, Debugger,
Cloud Synthesis

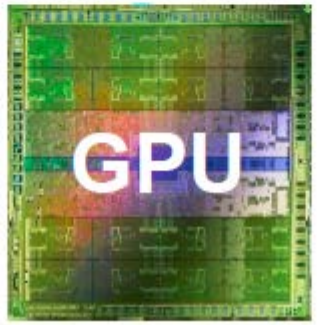


programmable accelerators (GPUs and FPGAs)

LIME

Toolchain





move fluidly between hardware and software, dynamically and adaptively.

LIME: Introduction

Lime = Java + Isolation + Abstract Parallelism

1. Prototype in standard Java (POO) high Abstraction

- Lime is a superset of Java (Extension)
- **Java-like Eclipse IDE (editors, debugger, navigation)**

2. Gentle, incremental migration to parallel Lime code

Lime language constructs restrict program to safe parallel structures

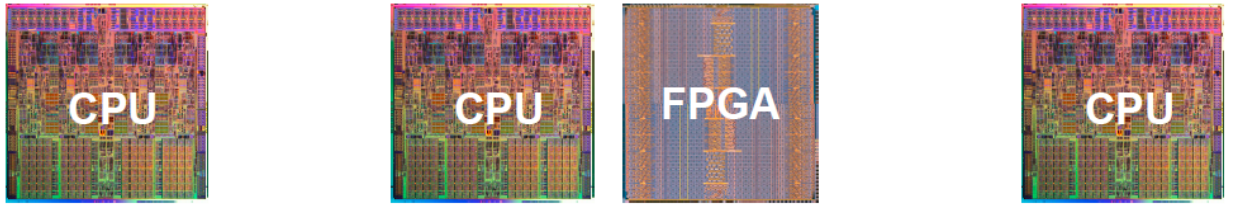
- Isolation : **ability to move computation**
- Immutability
- Safe Deterministic Parallelism

Software engineer can implement complex hardware in Lime

High-Level Java-Derived Language

```
public static local int foobar(int a) {
    return 2*a;
}
```

```
source() => ([ task new BlokusPlayer().nextMove ]) =>  sink()
```



References

- ***Website:*** JPEG Compression. Wikipedia.org
  https://en.wikipedia.org/wiki/JPEG
- ***Website:*** Liquid Metal IBM
  http://researcher.watson.ibm.com/researcher/view_group.php?id=122
- ***Paper:*** Lime: a Java-Compatible and Synthesizable Language for Heterogeneous Architectures, IBM Research
  http://www.cl.cam.ac.uk/research/srg/han/ACS-P35/readinglist/bacon-lime-p89-auerbach.pdf.
- ***Presentation:*** The Liquid Metal Blokus Experiment, Stephen Fink (IBM Research)
  http://researcher.ibm.com/researcher/files/us-rabbah/plday13-fink.pdf.