Controlling LED chains with FPGA

Author : Daniil Ignatev

TECHNISCHE UNIVERSITÄT
ILMENAU

Introduction

- LED chains:
  - Controlled by a serial protocol.
  - Can be controlled by software or hardware.
- Intellectual property (IP) core design:
  - Zynq7000 is the selected hardware platform.
  - Two approaches were studied: hardware description language (HDL) and high-level synthesis (HLS).

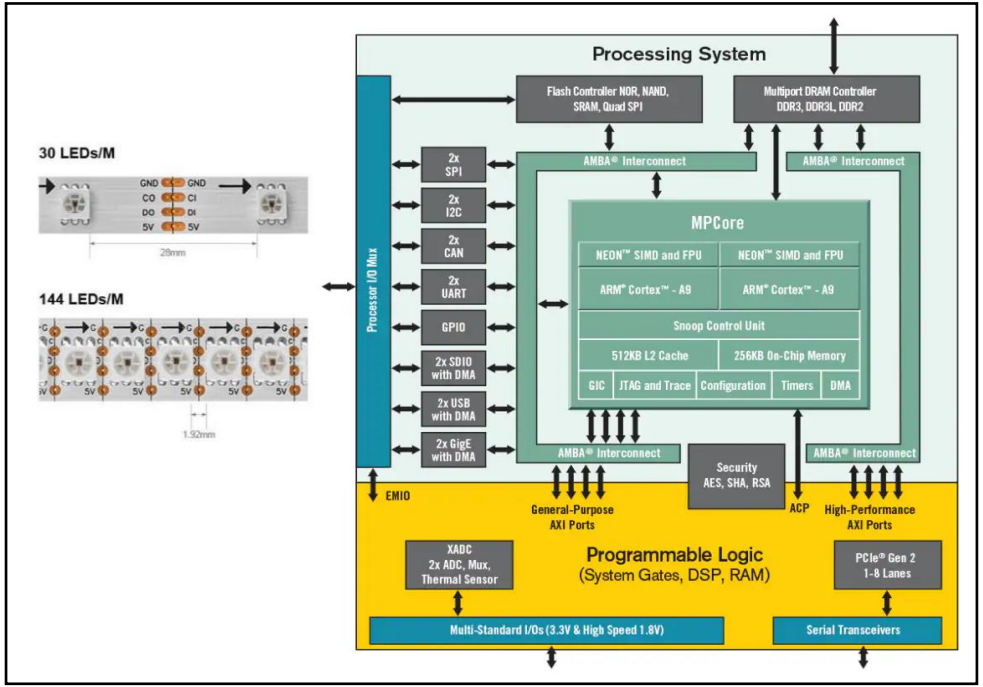


Fig. 1. LED chains and Zynq7000.

Task Description

- SK9822-compatible controller must be developed and documented.
- SW developers use memory-mapped interface to exchange data with CPU.
- HW developers integrate controller into the system.
- The final IP core will consist of the most effective approaches identified during the study.

Software Developer	**Hardware Developer**
Full color switching	Adjust LED number
Binary color switching	Limit brightness
Global brightness control	Set default brightness
Individual brightness control	
Select transmission flow	
Start transmission	

Table. 1. Use cases.

Research Methodology

Research Directions

- Design approaches
  - HDL (Verilog / System Verilog)
  - HLS (C / C++)
- Performance evaluation
  - Controller framerate
  - System framerate
- Measure resources footprint
  - Look-Up Tables
  - Flip-Flops
- Estimate scalability
  - Vertical
  - Horizontal

Research Methodology

Controller Framerate Formula

$$F_c(CLK, LEDs) = \frac{CLK}{c_{start} + c_{data} \cdot LEDs + c_{end}}, \tag{1}$$

where:

F_c = controller's framerate.
CLK = controller's clock frequency.
$LEDs$ = number of LEDs in the chain.
c_{start} = clock cycles number to transmit the start part of a frame.
c_{data} = clock cycles number to transmit a data part of a frame of a single LED.
c_{end} = clock cycles number to transmit the end part of a frame.

Controller Design

Internal Organization

- SK9822 controller is wrapped in AXI interface.
- SPI and Bytes Transmitter instances are optimized.
- Inheritance should be replaced by multiplexing.
- LED should be replaced by signals.

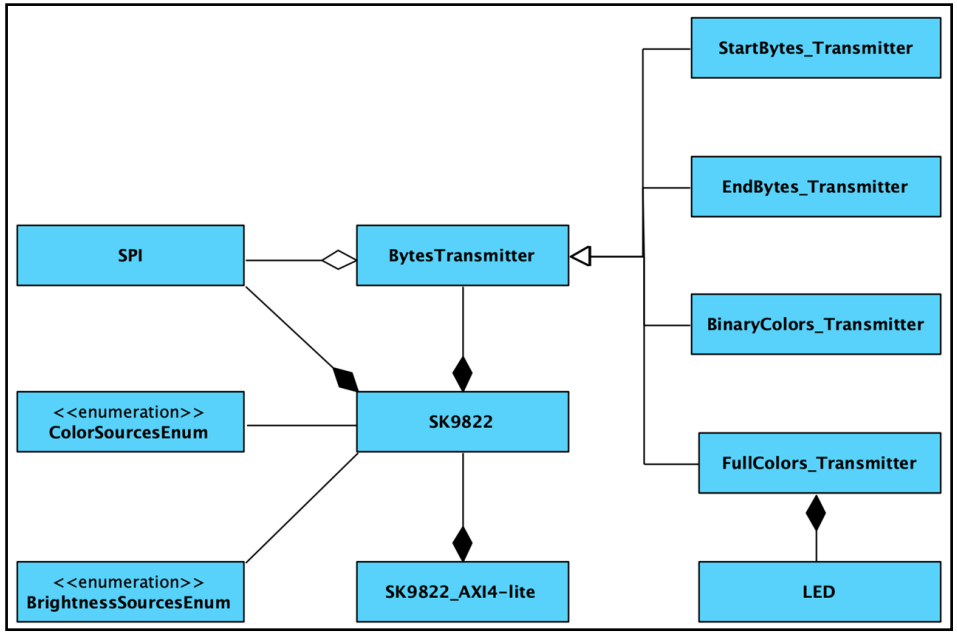


Fig. 2. Object-oriented model of the controller.

Controller Design

Runtime Scenarios

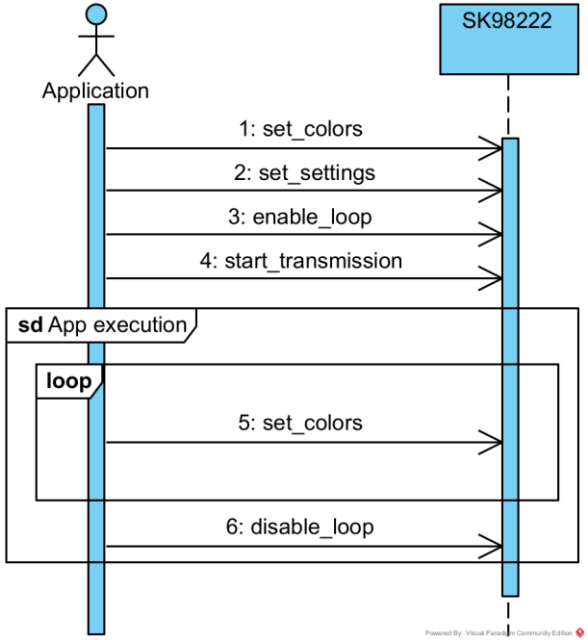


Fig. 3. Cyclic scenario.

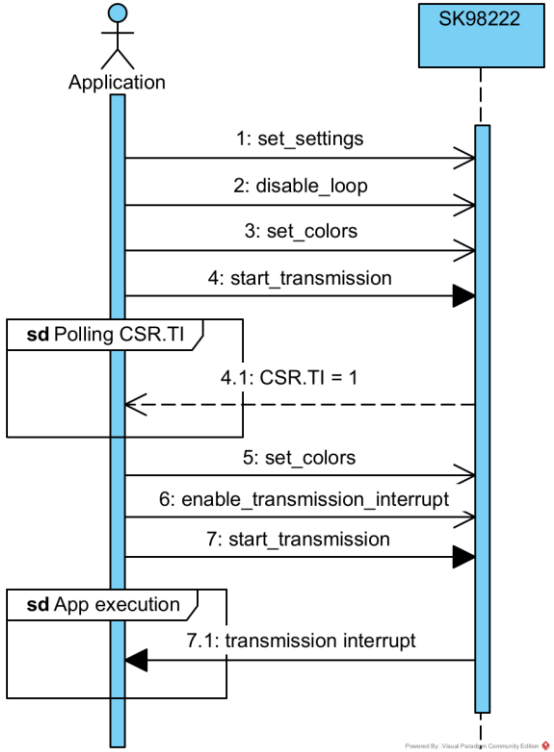


Fig. 4. Acyclic scenario.

Controller Design

Synchronous Start

- Synchronous start is a feature.
- SCLK is shared.
- I/O are optimized.
- Main controller must be selected.

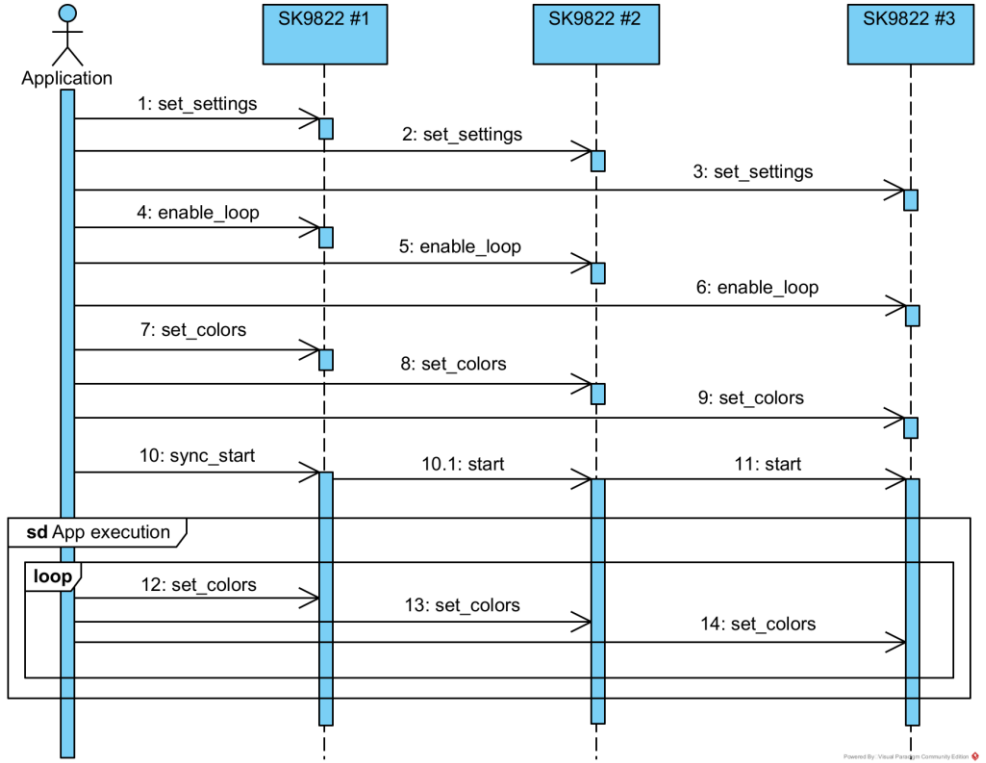


Fig. 5. Cyclic scenario + synchronous start.

Study on high level synthesis (HLS)

Introduction

- C/C++ functions are translated into RTL.
- Provides a higher level of abstraction.
- Applicability is a question.
- Results comparison is the motivation.

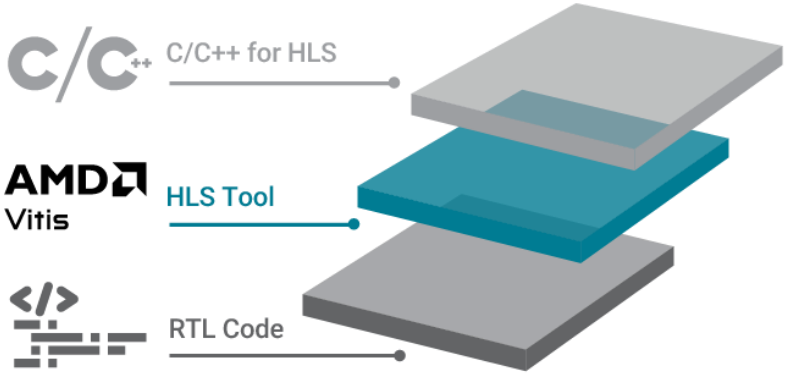


Fig. 6. AMD Vitis HLS overview.

Study on high level synthesis (HLS)

Findings

Architecture features:

- Concurrent C / C++ code.
- State machines designed automatically.
- SPI implementation is a challenge.
- Tight timing increases development cost.

Proposed techniques:

- “#pragma HLS inline off”.
- “#pragma HLS protocol” blocks.
- Call “ap_wait()” delay function.
- volatile output pointers in the top.

Signal Diagrams

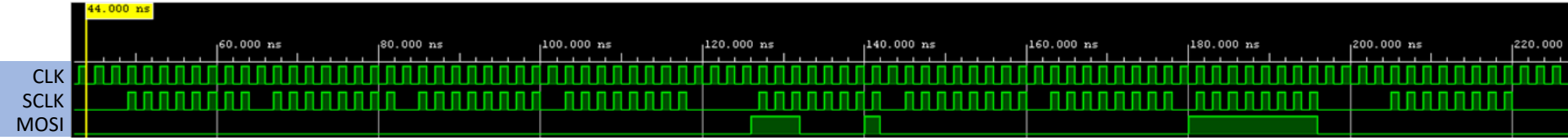


Fig. 7. HDL IP signal diagram.

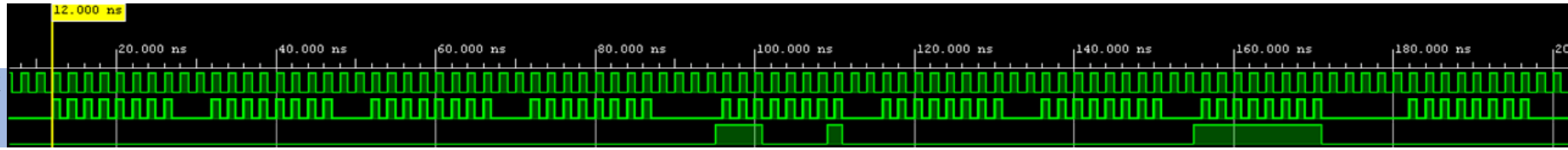
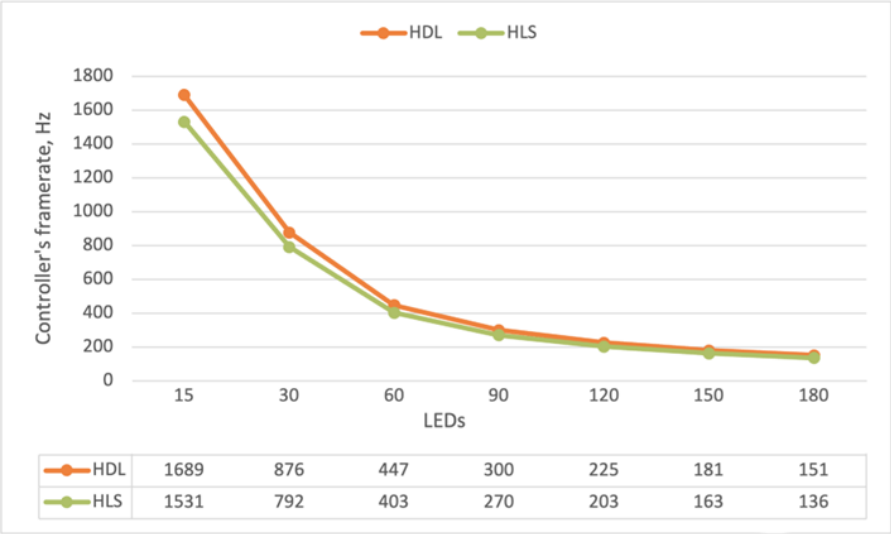


Fig. 8. HLS IP signal diagram.

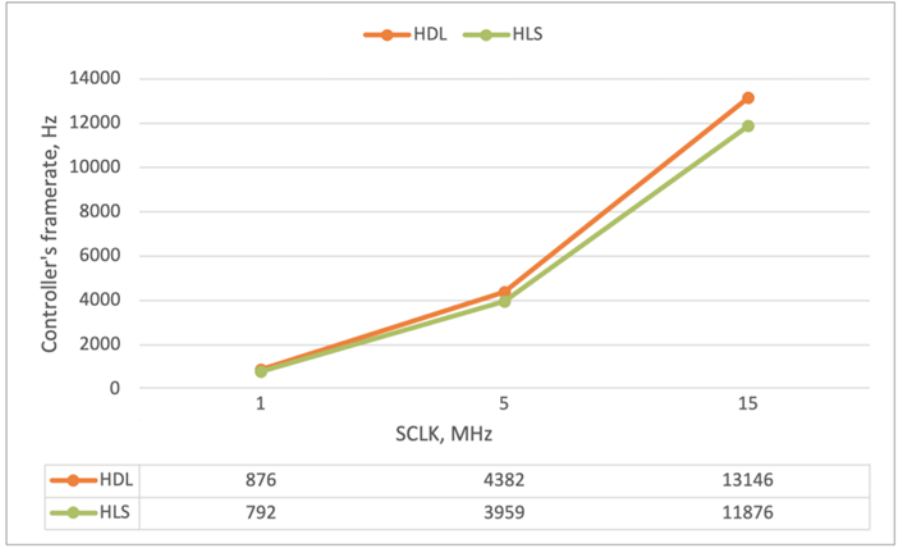
Results

Comparison of FPGA controllers' performance



HDL	1689	876	447	300	225	181	151
HLS	1531	792	403	270	203	163	136

Fig. 9. Fc(1MHz, LEDs)



HDL	876	4382	13146
HLS	792	3959	11876

Fig. 10 Fc(SCLK, 30)

Results

Comparison of FPGA Resources Footprints

	HDL	HLS	AXI4-lite (30LEDs)	Occupied by other blocks	Available in Zynq7020
Lookup tables	375	921	559	691	53200
Flip-flops	38	148	1350	737	106400

Table 2. FPGA resources utilization.

Summary and Outlook

- Requested SK9822-compatible IP cores were developed.
- A study was conducted on HDL and HLS approaches.
- Resulting IP may be used for scalability-sensitive applications.
- HLS has room for improvement.

Appendix

System Framerate Formula

$$F_s(F_c, t_{system}) = \frac{1}{\frac{1}{F_c} + t_{system}}, \tag{2}$$

where:

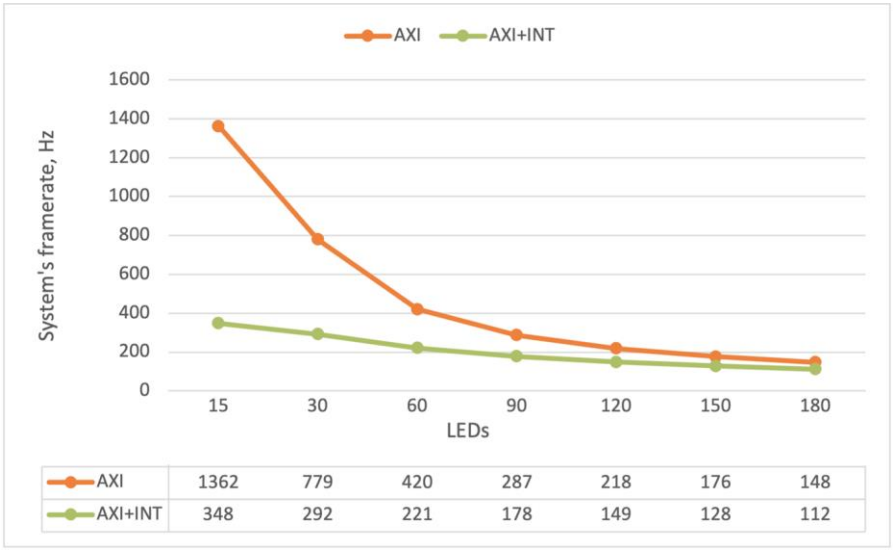
F_s = system's framerate.

F_c = frame transmission time, can be measured or calculated using the formula 1.

t_{system} = delay caused by system (CPU and AXI interface).

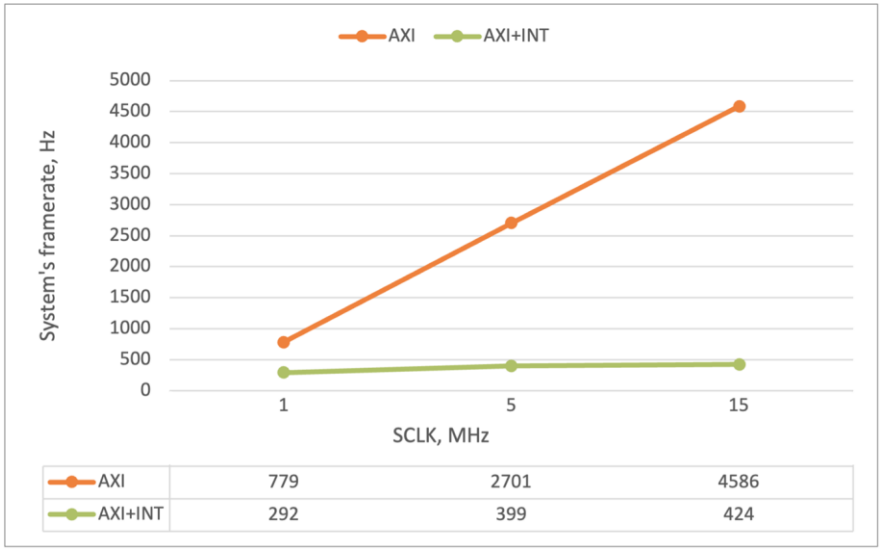
Appendix

Comparison of FPGA systems' performance



AXI	1362	779	420	287	218	176	148
AXI+INT	348	292	221	178	149	128	112

Fig. 11. FPGA Fs(1MHz, LEDs)



AXI	779	2701	4586
AXI+INT	292	399	424

Fig. 12. FPGA Fs(CLK, 30)

Appendix

Arduino UNO performance

GPIO	1449	833	441	303	231	184	154
SPI	1471	852	435	296	225	181	151

Fig. 13. MCU Fs(1MHz, LEDs)

GPIO	833	833
SPI	852	2079

Fig. 14. MCU Fs(CLK, 30)