TECHNISCHE UNIVERSITÄT DRESDEN

Chair for Embedded Systems

Faculty of Computer Science Institute of Computer Engineering, Chair for Embedded Systems

REGISTER ALLOCATION FOR HIGH-LEVEL SYNTHESIS OF HARDWARE ACCELERATORS TARGETING FPGAS

Gerald Hempel, Christian Hochberger, Jan Hoyer, Thilo Pionteck

Darmstadt, 12 July 2013

Outline

- Motivation
- Design Flow
- Register Allocation
- Results & Discussion
- Summary

Motivation

- FPGAs suitable platform for application-specific designs
- Predominant concept → standard processor IP-core in combination with application-specific accelerators
- Goal: Automatic mapping of high level application code into HW
  - Synthesis result is a combination of HLS and vendor synthesis tools (XST, Synopsis, Altera Synthesizer)

Motivation

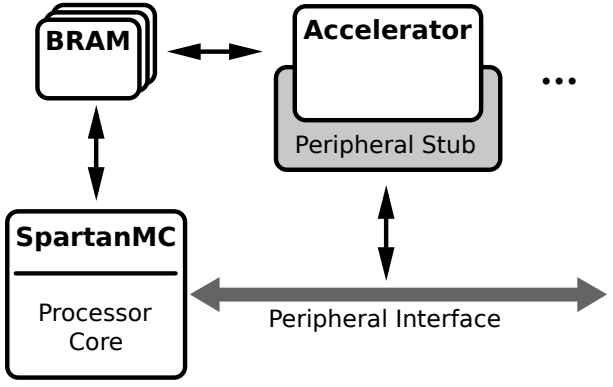
- FPGAs suitable platform for application-specific designs
- Predominant concept → standard processor IP-core in combination with application-specific accelerators
- Goal: Automatic mapping of high level application code into HW
  - Synthesis result is a combination of HLS and vendor synthesis tools (XST, Synopsis, Altera Synthesizer)

How much effort is required in HLS optimization?

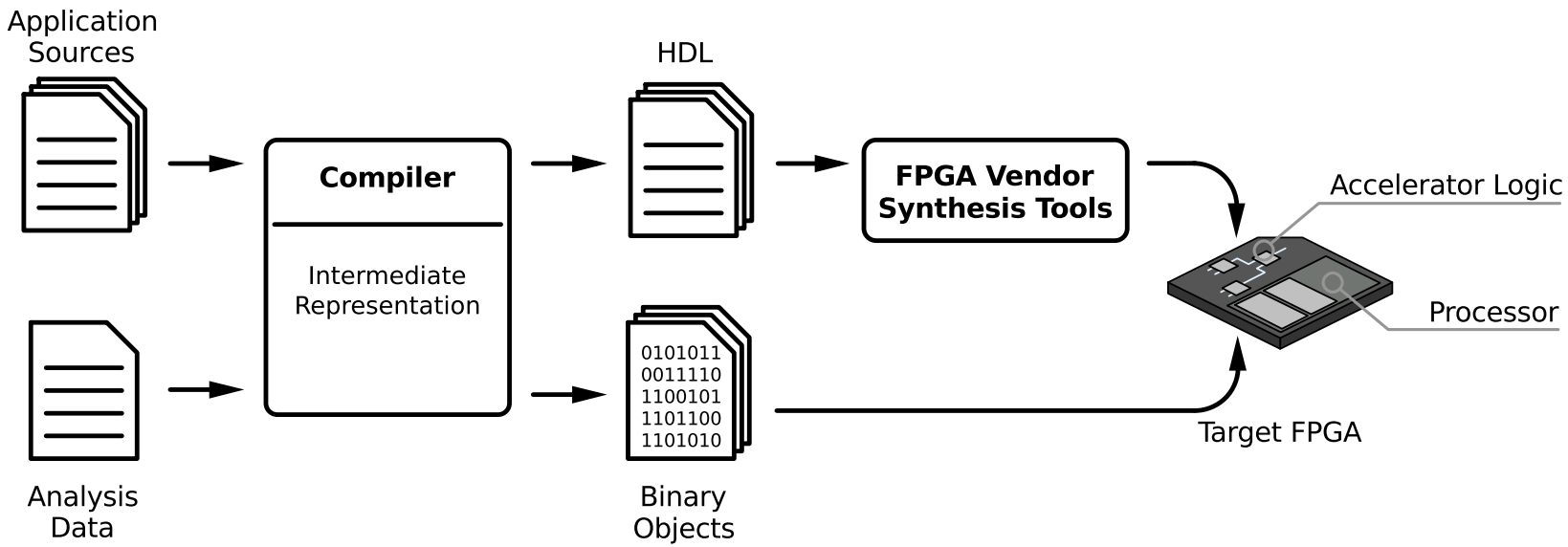
- Evaluation of several register allocation strategies for FPGA based accelerators on Spartan 6 and Artix 7
- Underlying synthesis tool: Xilinx XST P.49d in ISE 14.4

SpartanMC Soft-Core and Kernel Interface

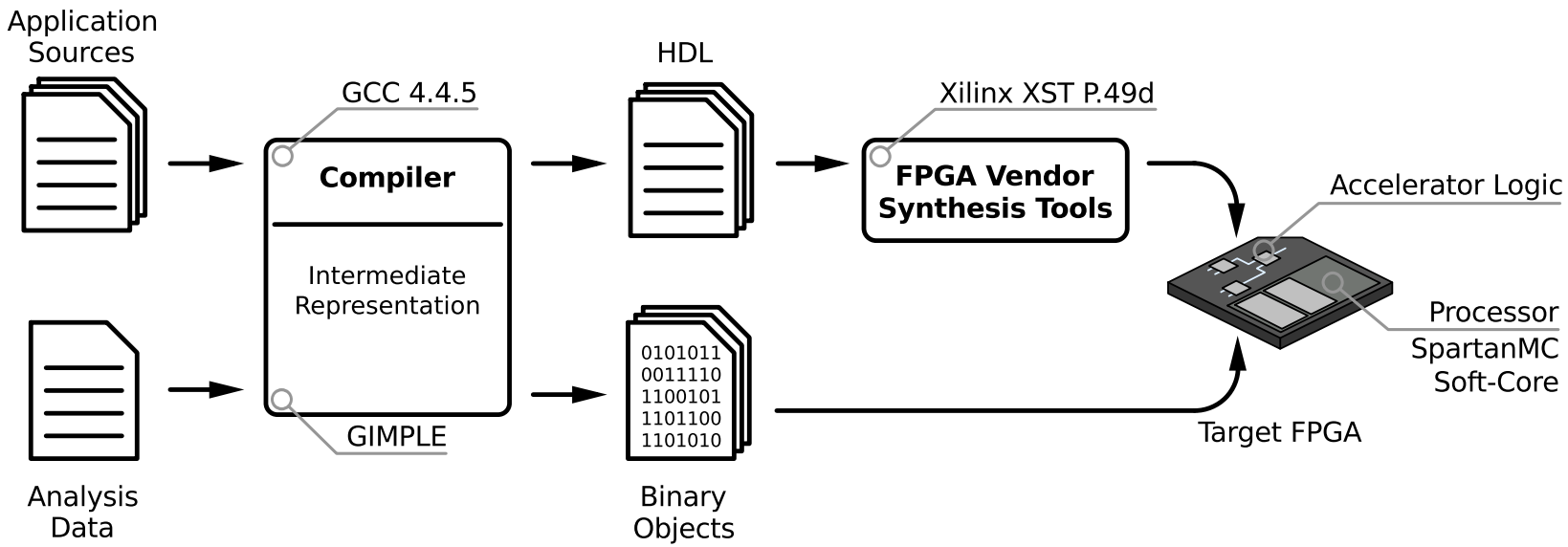
- SpartanMC processor soft-core for software execution
- Program and data stored in local BRAM
- HW accelerators treated as peripherals
- Peripheral stub used as wrapper for accelerator
- Peripheral stub provides access to memory and peripheral bus
  - Parameters transfered via peripheral bus during kernel startup
  - Direct BRAM accesses (triggered by pointers and arrays) during kernel execution



Typical Workflow

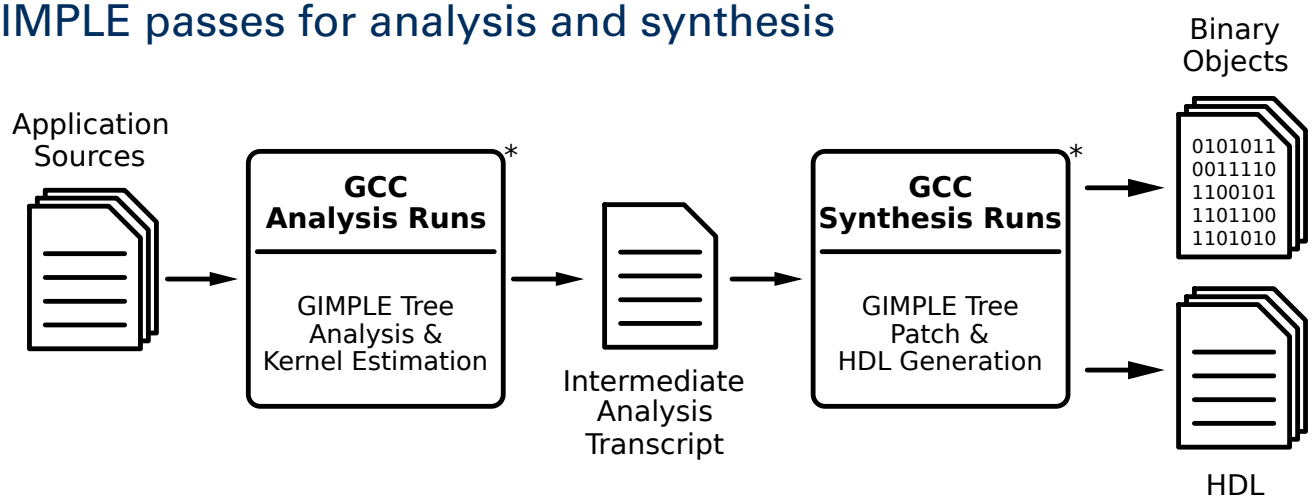


Typical Workflow



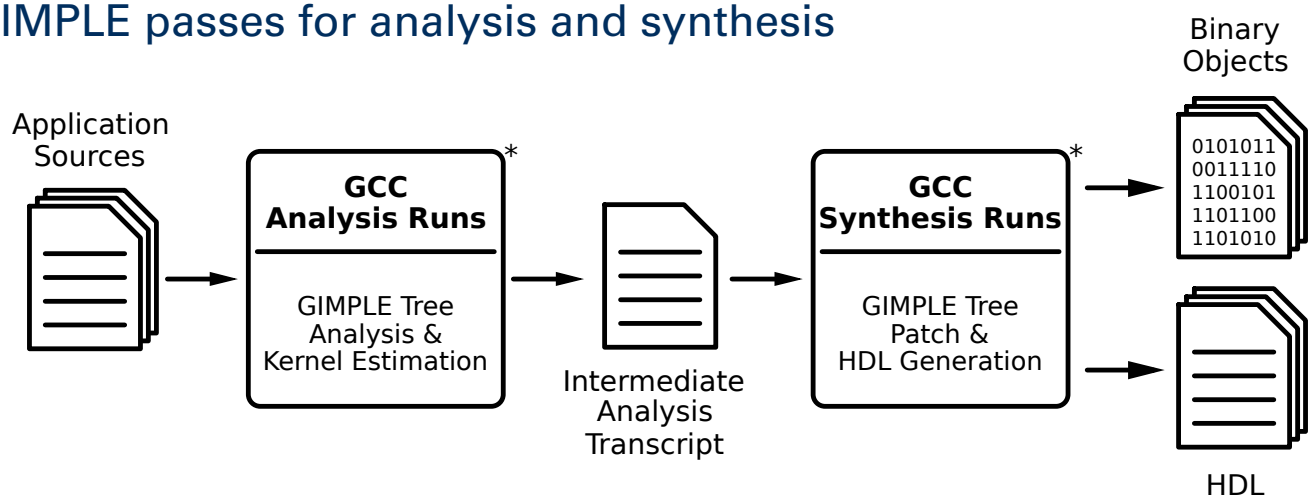
Kernel Extraction using GCC

- GIMPLE passes for analysis and synthesis



Kernel Extraction using GCC

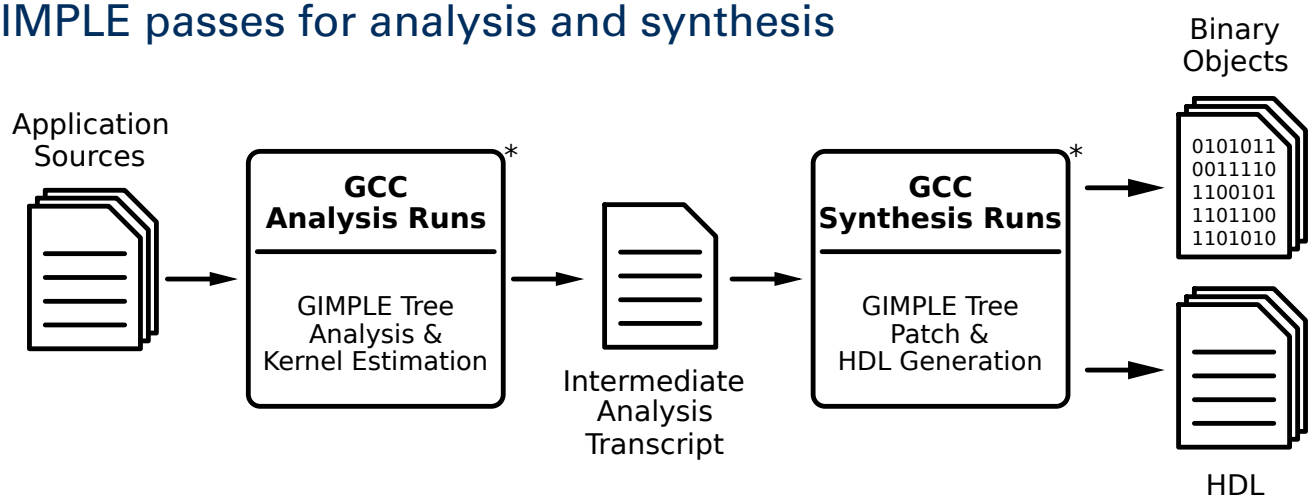
- GIMPLE passes for analysis and synthesis



- Analysis:
  - Finds most worthy loop
  - Omits functions calls (inlined functions only)

Kernel Extraction using GCC

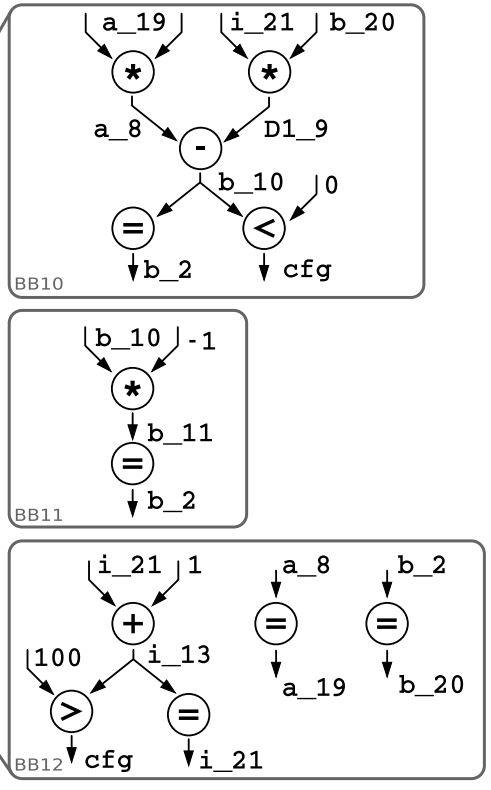
- GIMPLE passes for analysis and synthesis



- Analysis:
  - Finds most worthy loop
  - Omits functions calls (inlined functions only)
- Synthesis:
  - Uses list scheduling
  - Performs high-level register allocation

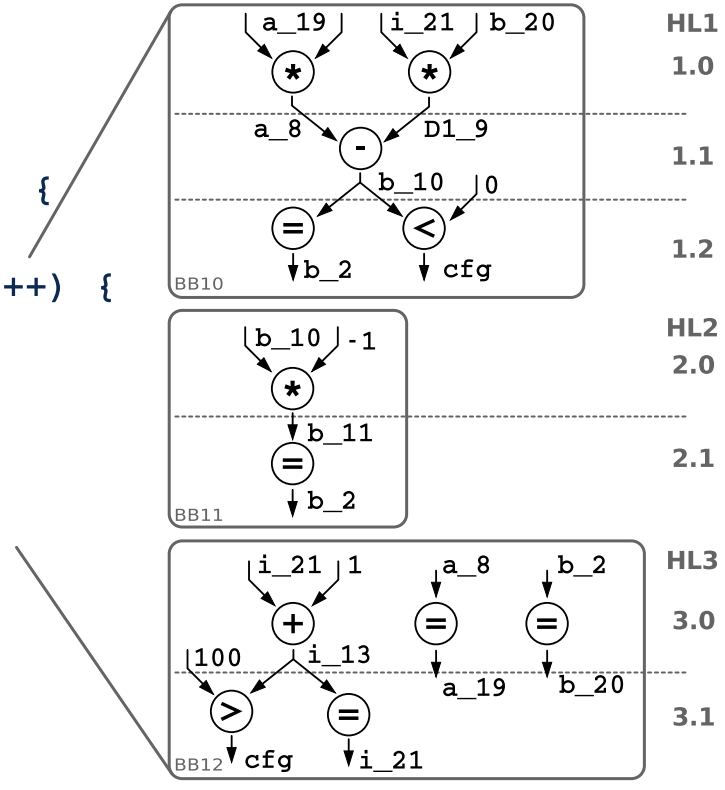
Register Allocation (Modified Left-Edge)

```
int foo (int a, int b) {
  int i;
    for (i=0; i<100; i++) {
      a = a * a;
      b = a - b * i;
      if (b < 0)
        b = b * -1;
    }
  return b;
}
```



Register Allocation (Modified Left-Edge)

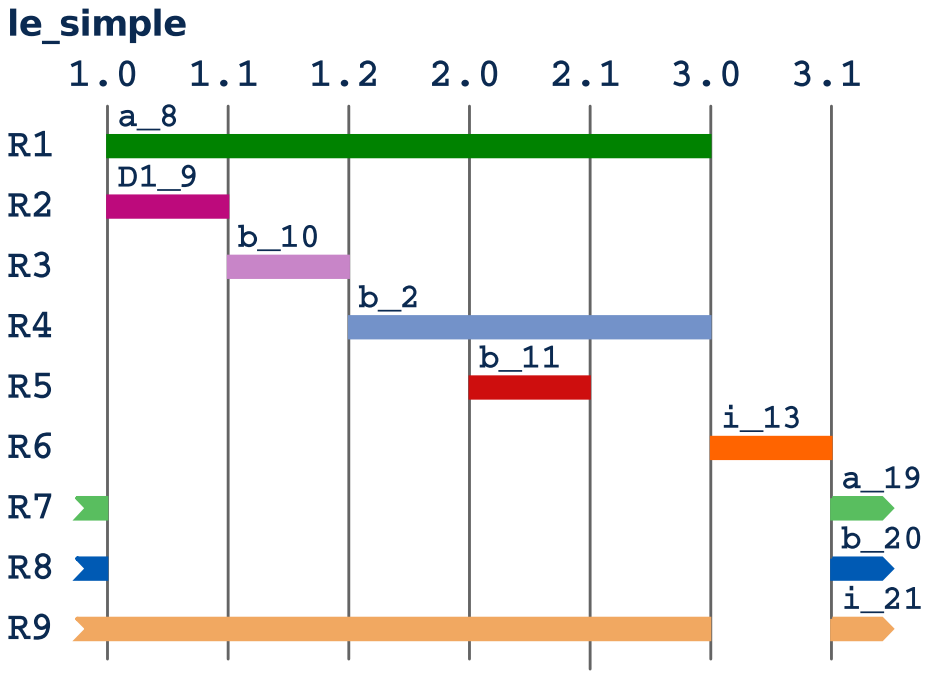
```
int foo (int a, int b) {
  int i;
    for (i=0; i<100; i++) {
      a = a * a;
      b = a - b * i;
      if (b < 0)
        b = b * -1;
    }
  return b;
}
```



Register Allocation Strategies

- **le_simple:** Assigns a register for each GIMPLE-variable

```
a_8:   1.0-3.0 (global)
D1_9:  1.0-1.1 (local)
b_10:  1.1-1.2 (local)
b_2:   1.2-3.0 (global)
b_11:  2.0-2.1 (local)
i_13:  3.0-3.1 (local)
a_19:  3.1-1.0 (global,loop)
b_20:  3.1-1.0 (global,loop)
i_21:  3.1-3.0 (global,loop)
```

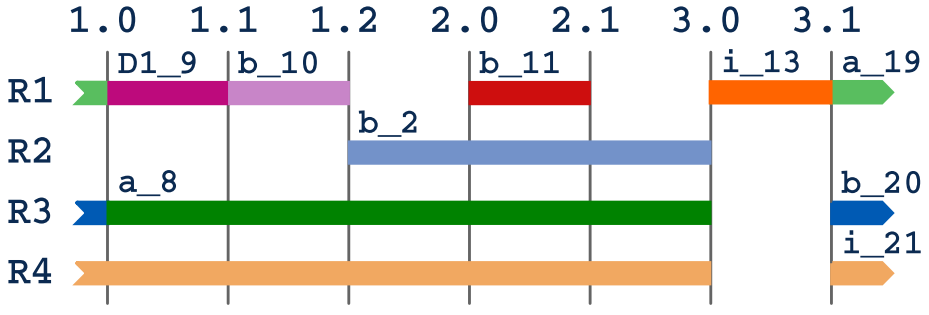


Register Allocation Strategies

- **le_full:** Minimize the number of registers

```
a_8:   1.0-3.0 (global)
D1_9:  1.0-1.1 (local)
b_10:  1.1-1.2 (local)
b_2:   1.2-3.0 (global)
b_11:  2.0-2.1 (local)
i_13:  3.0-3.1 (local)
a_19:  3.1-1.0 (global,loop)
b_20:  3.1-1.0 (global,loop)
i_21:  3.1-3.0 (global,loop)
```

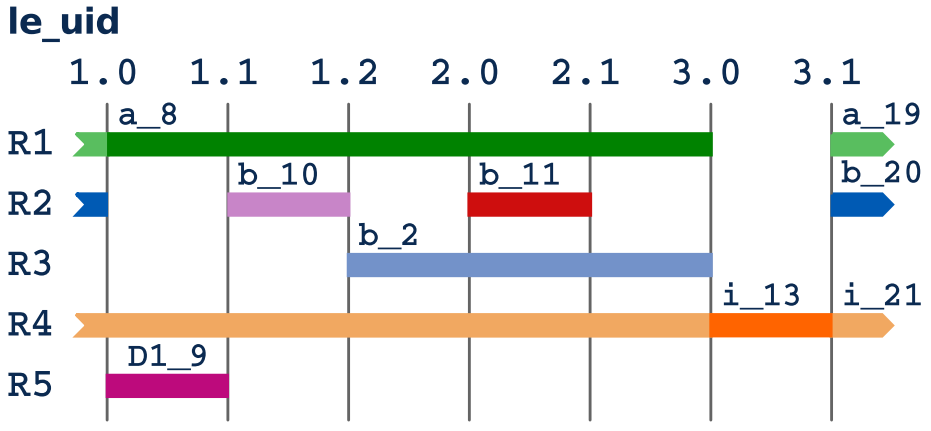
le_full



Register Allocation Strategies

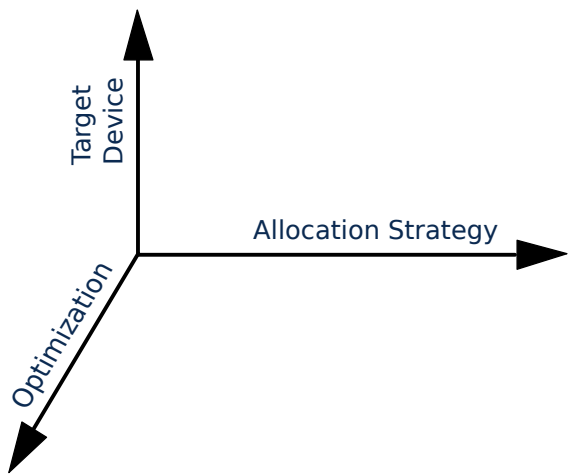
- **le_uid:** Maps variables with identical GIMPLE-ID to one register

```
a_8:   1.0-3.0 (global)
D1_9:  1.0-1.1 (local)
b_10:  1.1-1.2 (local)
b_2:   1.2-3.0 (global)
b_11:  2.0-2.1 (local)
i_13:  3.0-3.1 (local)
a_19:  3.1-1.0 (global,loop)
b_20:  3.1-1.0 (global,loop)
i_21:  3.1-3.0 (global,loop)
```

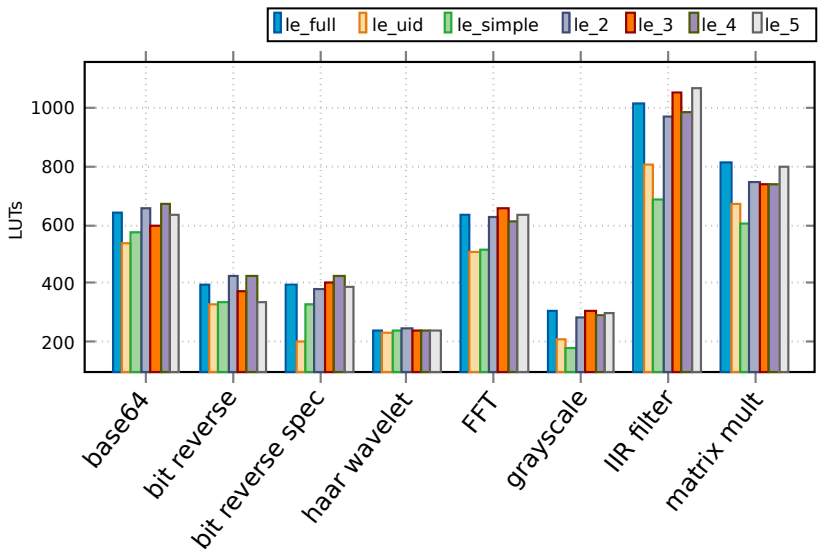


Benchmarks

- Typical algorithms for embedded systems:
  - base64 encoder, bitreverse, FFT, grayscale filter, IIR filter, haar wavelet transformation, matrix multiplication
- Kernels were generated automatically for compute intensive parts
- Parameter sweep:
  - *Area* or *speed* optimization
  - *Spartan 6* or *Artix 7* device
  - Allocation strategy
    (le_full, le_simple, le_2, le_3, le_4, le_5, le_uid)
  - 8 benchmarks
- 224 test cases

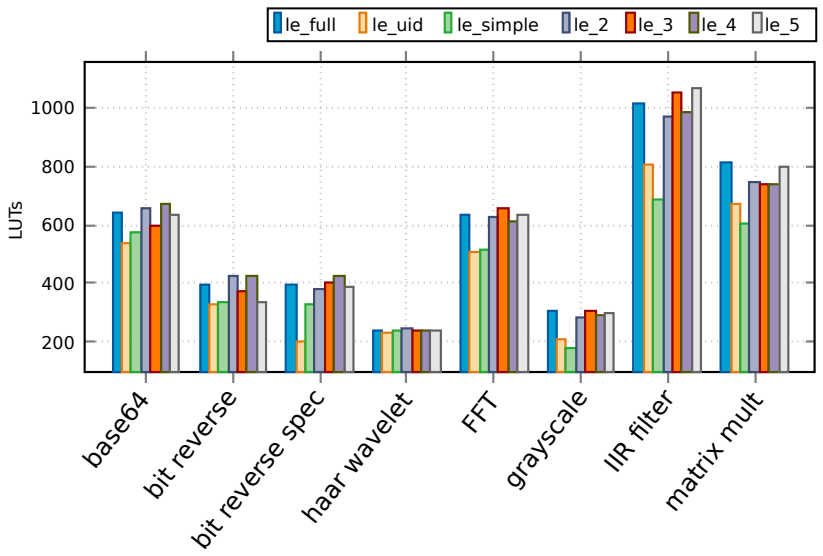


Artix 7 Resource Consumption



Used LUTs (optimized for area)

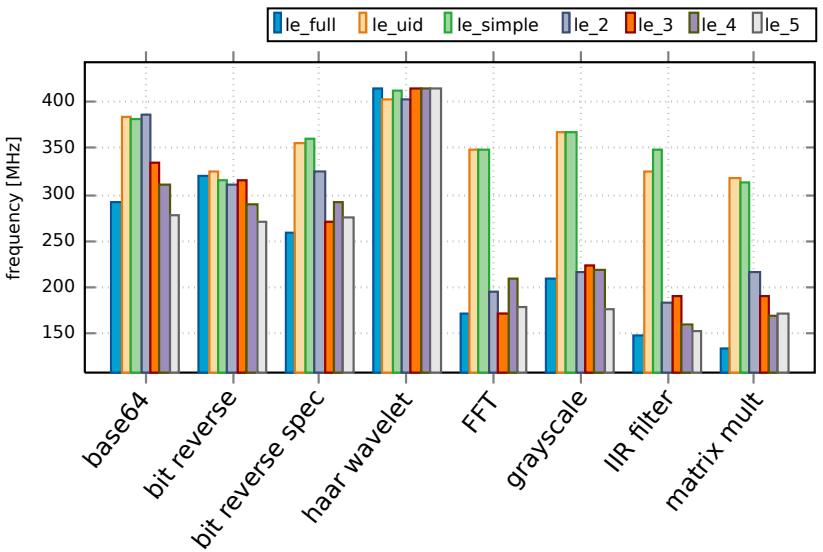
Artix 7 Resource Consumption



Used LUTs (optimized for area)

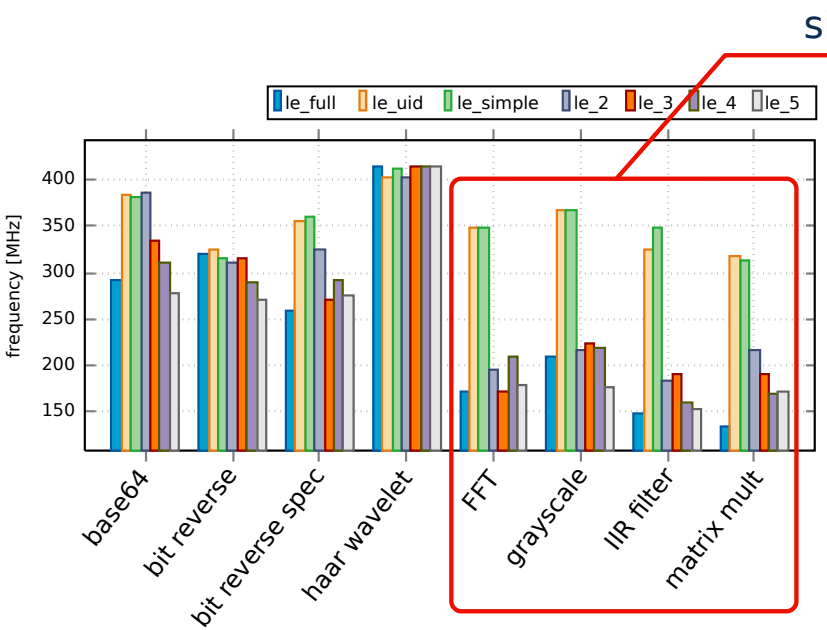
- Allocation strategies show little effect on resource consumption
- Best results for *le_simple* and *le_uid*

Artix 7 Frequency Results



Achievable clock frequency
(optimized for area)

Artix 7 Frequency Results



Achievable clock frequency (optimized for area)

- Results for *le_2, le_3, le_4, le_5* hard to predict (s. haar wavelet, bit reverse and base 64 encoder)
- FFT, grayscale filter, IIR filter and matrix multiplication → negative effect for all allocation strategies except *le_uid* and *le_simple*

What is the difference?

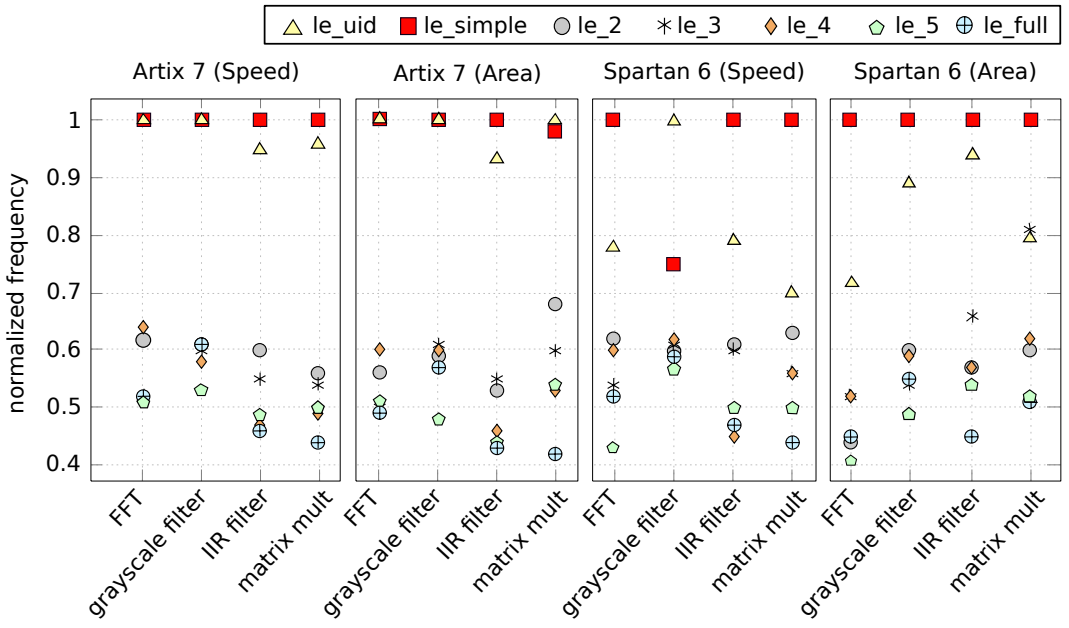
Benchmarks	#GIMPLE variables	#basic blocks	#branches	#ALU operations	#MUL operations
base64 encode	50	3	1	21	0
bitreverse	62	22	11	61	0
bitreverse (spec.)	62	6	3	61	0
haar wavelet	14	3	1	7	0
FFT	51	6	2	22	4
grayscale filter	29	3	1	17	2
IIR filter	79	6	2	29	11
matrix mult	59	3	1	25	8

What is the difference?

Benchmarks	#GIMPLE variables	#basic blocks	#branches	#ALU operations	#MUL operations
base64 encode	50	3	1	21	0
bitreverse	62	22	11	61	0
bitreverse (spec.)	62	6	3	61	0
haar wavelet	14	3	1	7	0
FFT	51	6	2	22	4
grayscale filter	29	3	1	17	2
IIR filter	79	6	2	29	11
matrix mult	59	3	1	25	8

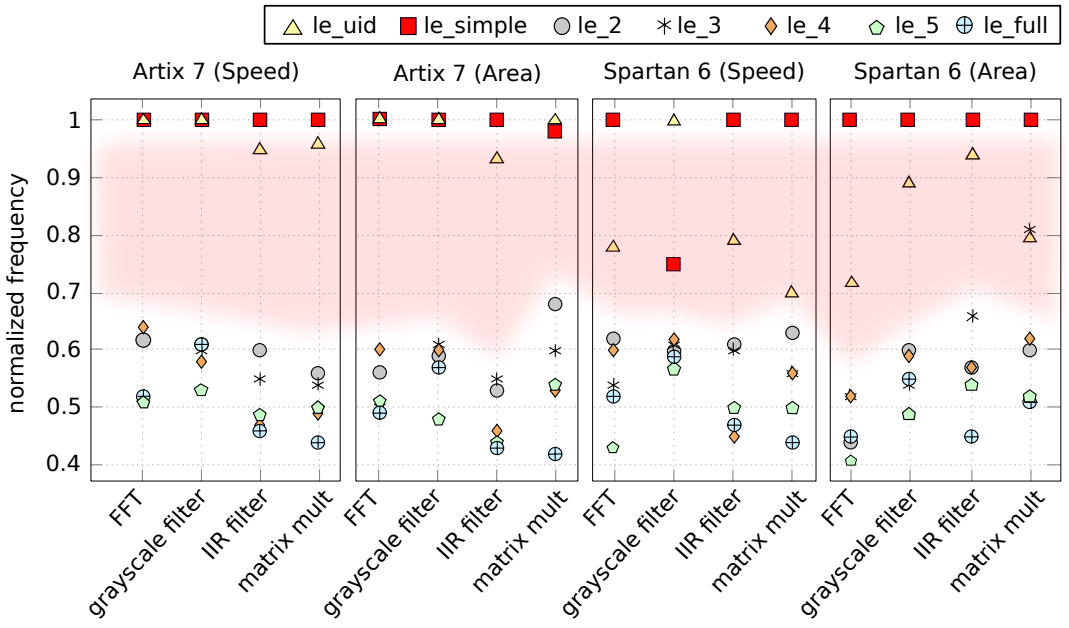
- Multiply-accumulate operation are mapped to sequential DSP primitives
- Multiplexer tree caused by variable swaps leads to performance degradations

Artix 7 normalized Frequency



- Achievable clock frequency optimized for area normalized to one

Artix 7 normalized Frequency



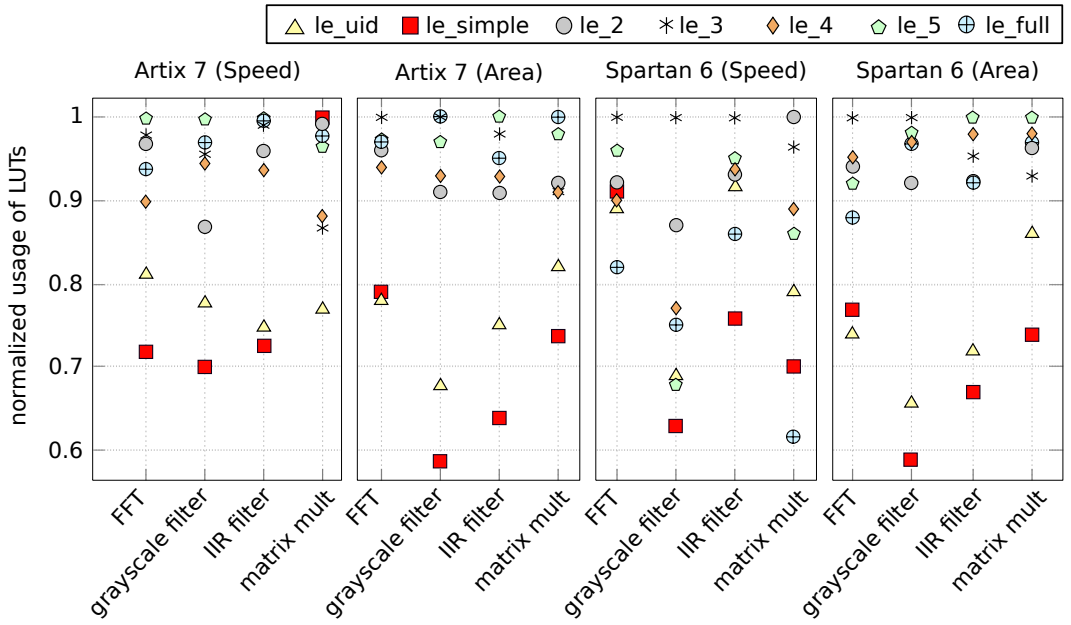
- Achievable clock frequency optimized for area normalized to one
- Gap of about 30% between *le_simple*, *le_uid* and more complex algorithms

Conclusion

For HLS targeting FPGAs:

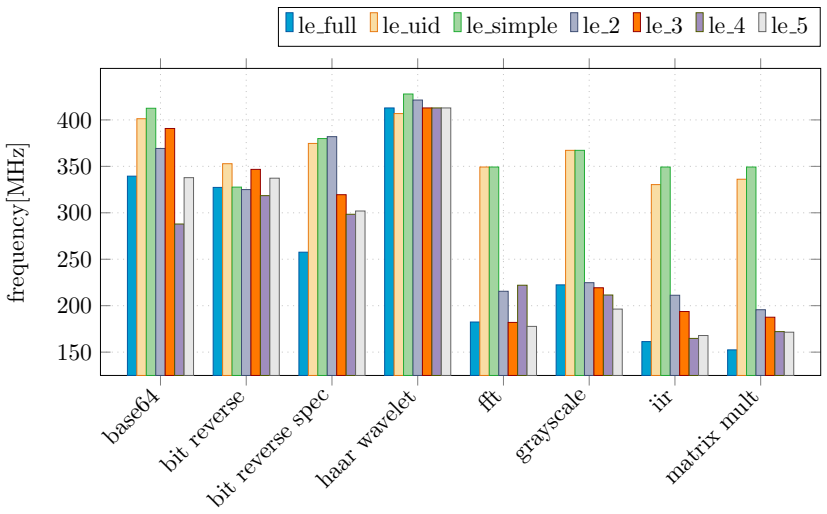
- Complex registers allocation strategies resulting in a reuse of a single register for many variables are not advisable
  - Little effect on area consumption
  - May lead to performance degradations (up to 30%)
- Naive allocation strategy gives most freedom to synthesis tool and mostly better results

Artix 7 normalized LUTs

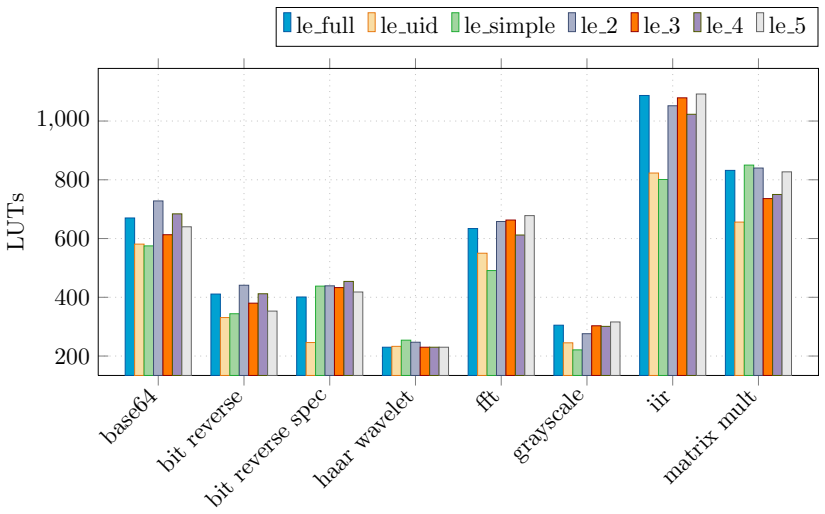


- A simple allocation strategy inclines a small resource footprint.

Artix 7 Frequency and LUTs (speed)

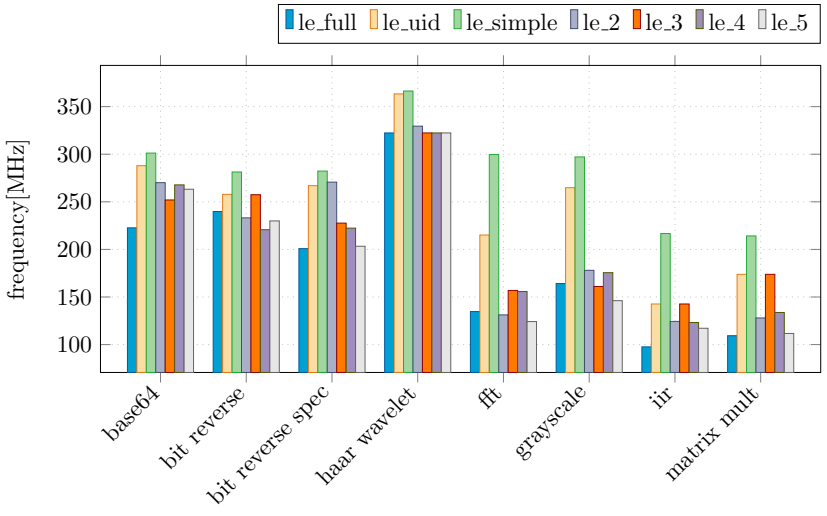


- Achievable clock frequency optimized for speed

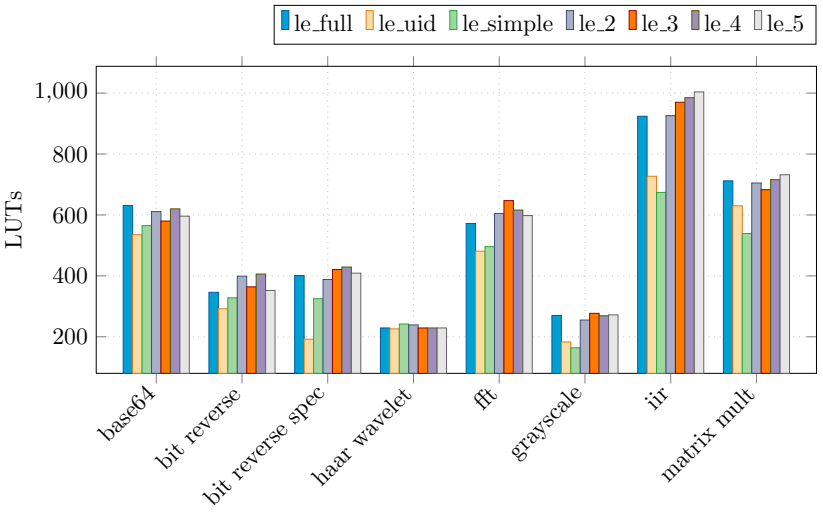


- Used LUTs optimized for speed

Spartan 6 Frequency and LUTs (area)

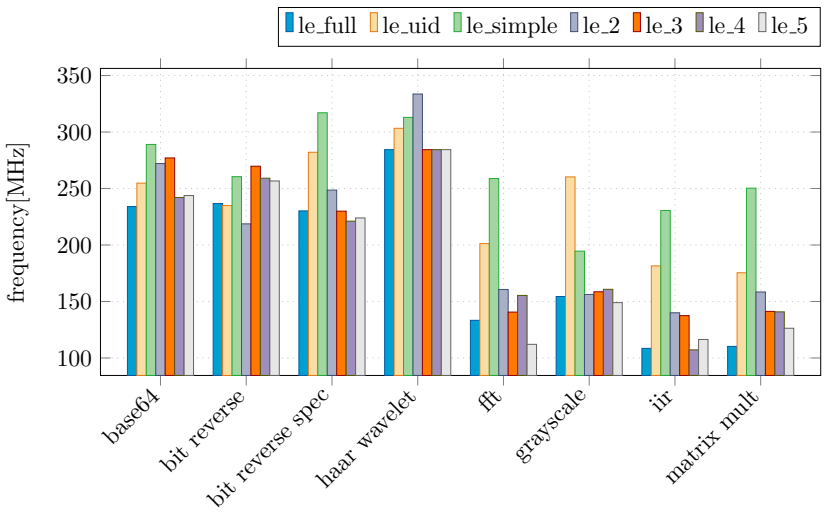


- Achievable clock frequency optimized for area

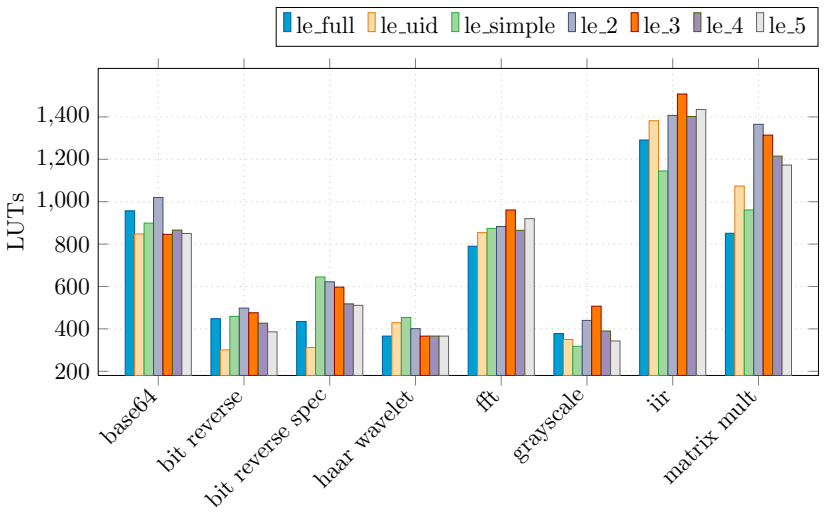


- Used LUTs optimized for area

Spartan 6 Frequency and LUTs (speed)



- Achievable clock frequency optimized for speed



- Used LUTs optimized for speed