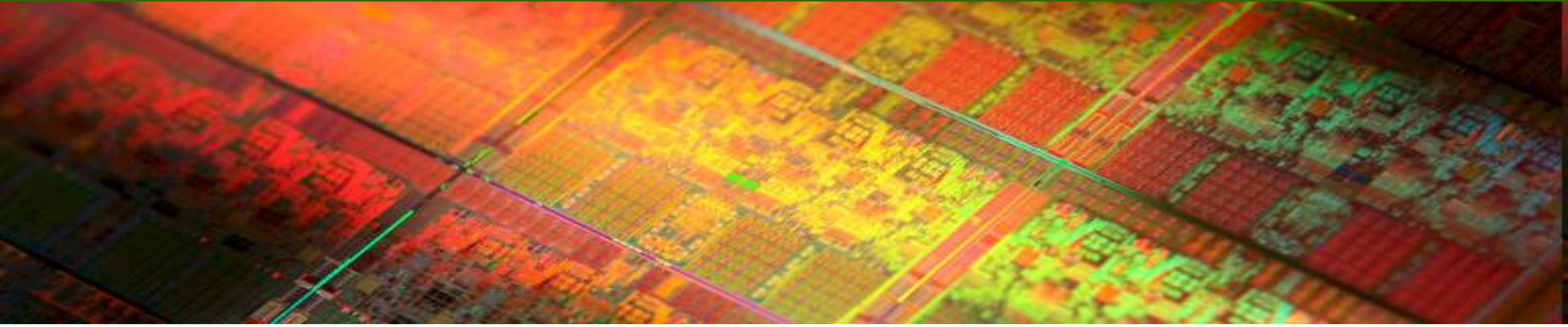
Queen Mary
University of London

EECS
School of Electronic Engineering
and Computer Science



Circuit Design and Analysis for Approximate Integer Format

OPRECOMP Summer of Code Presentation 4th September at
Architectures and Algorithms for Energy-Efficient IoT and HPC Applications
(Perugia, Sept 2019)

Updated: 4 September 2019

Matthew Tang (matthew.tang@qmul.ac.uk)

Overview

- Gao's Approximate Integer Format (AIF)
- Algorithm for AIF addition
- AIF Adder design
- Experimental results
- Pipelined version
- Conclusion / future work
- Q & A

Approximate Integer Format (AIF)

- Gao et al "*A Novel Data Format for Approximate Arithmetic Computing*", ASPDAC, 2017.
- Aim to enable approximate arithmetic at ISA level by providing extra hardware called AIF modules (i.e. checkers, adders, multipliers)
- A data word is partitioned into (data) blocks.
  - The block size is 4 bits (as in the paper).
  - A block is valid if it is non-zero.
  - Sentinel (ST) bits: mark the position of the largest valid data blocks

AIF: Examples

16-bit word, 1 st 3 data blocks, block size k = 4

ST	D_2	D_1	D_0

e.g. 6128 = 0x184A = 0001 1000 0100 1010

approximates as

1111 0001 1000 0100

32-bit word, 2 st 6 data blocks, block size k = 4

ST_1	ST_0	D_5	D_4	D_3	D_2	D_1	D_0

e.g. 5109881 = 0x004DF879

approximates as

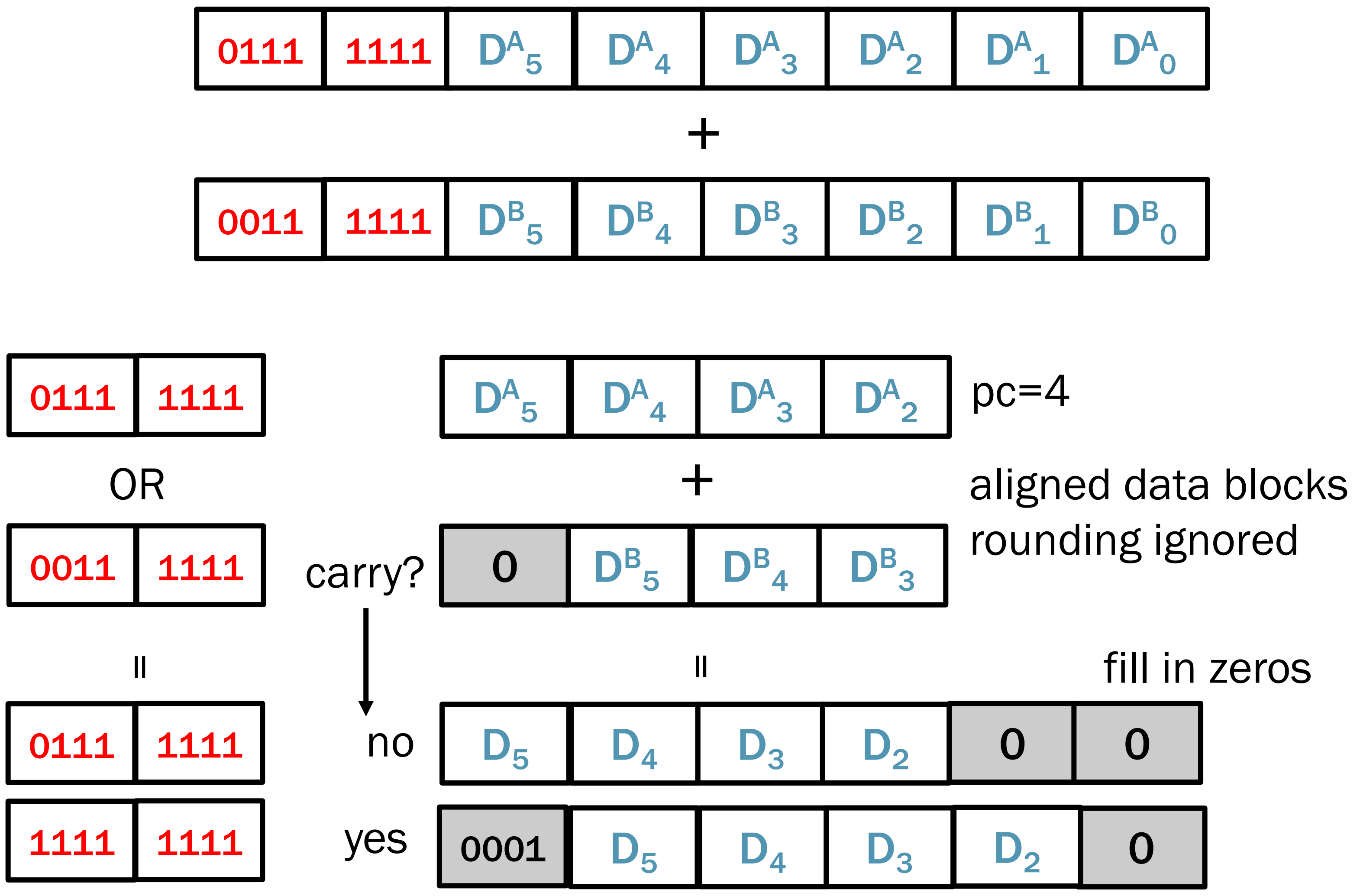
0011 1111 0100 1101 1111 1000 0111 1001

AIF: Addition

- The format is accurate when the integer is small.
  32 bits: accurate $< 2^{24}$; approximate: $2^{24} <= x < 2^{32}$
- In arithmetic operations, further approximation is introduced by a precision control (pc) parameter, e.g. pc = 4 for 6 data blocks.
  Reduce bit width for adder and thus area and power.
- Overheads: to expand compressed data, to maintain sentinel blocks, and to align the sum when there is a carry out.

Are these overheads justified?
Could that be compensated by area and power improvement, given a reasonably efficient implementation?

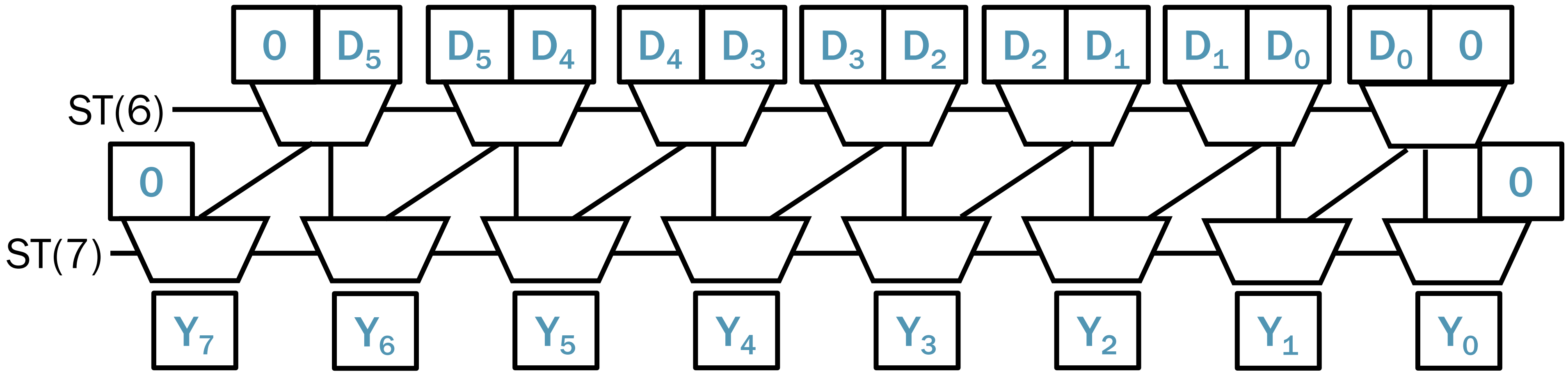
AIF Addition Algorithm



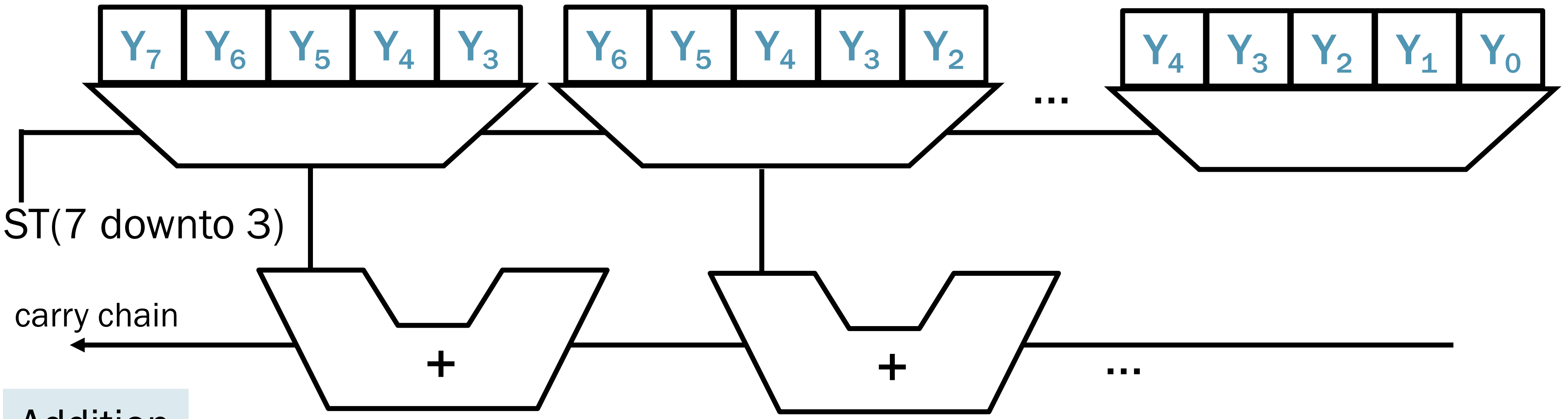
0111 | 1111 | D^A_5 | D^A_4 | D^A_3 | D^A_2 | D^A_1 | D^A_0
+
0011 | 1111 | D^B_5 | D^B_4 | D^B_3 | D^B_2 | D^B_1 | D^B_0
0111 | 1111
OR
0011 | 1111
=
D^A_5 | D^A_4 | D^A_3 | D^A_2
pc=4
+
aligned data blocks
rounding ignored
carry?
0 | D^B_5 | D^B_4 | D^B_3
=
fill in zeros
no
0111 | 1111
D_5 | D_4 | D_3 | D_2 | 0 | 0
yes
1111 | 1111
0001 | D_5 | D_4 | D_3 | D_2 | 0

AIF Adder: Design (1)

Decompression: Mux arranged like a barrel shifter

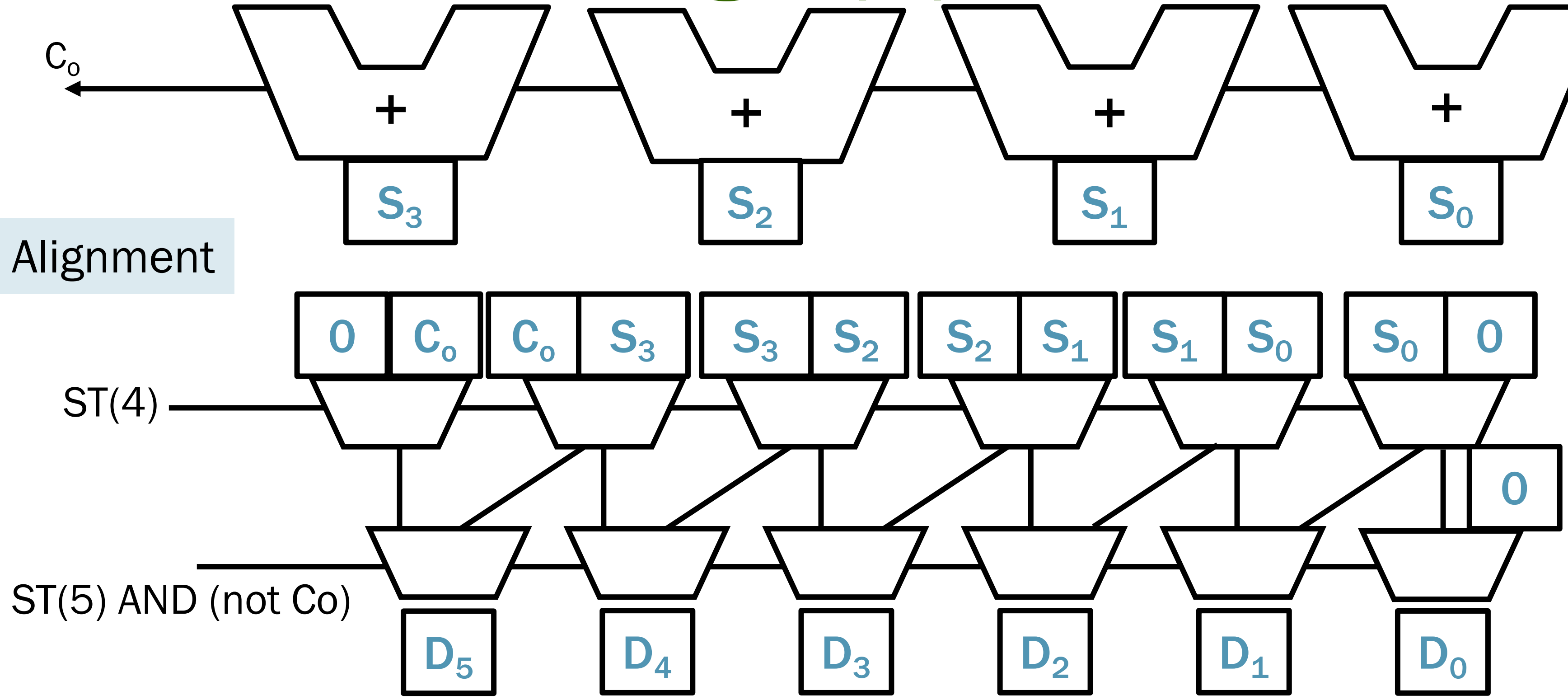


Selection: pick the right blocks


Y_7 Y_6 Y_5 Y_4 Y_3
Y_6 Y_5 Y_4 Y_3 Y_2
...
Y_4 Y_3 Y_2 Y_1 Y_0
ST(7 downto 3)
carry chain
+
+
...

Addition

AIF Adder: Design (2)



Generate new ST blocks

ST(7)	ST(6)	ST(5)	ST(4)	ST(3)	ST(2)	ST(1)	ST(0)

OR

C(3)	C(3)	C(3)	C(3)	C(2)	C(1)	C(0)	1

AND

ST(6)	ST(5)	ST(4)

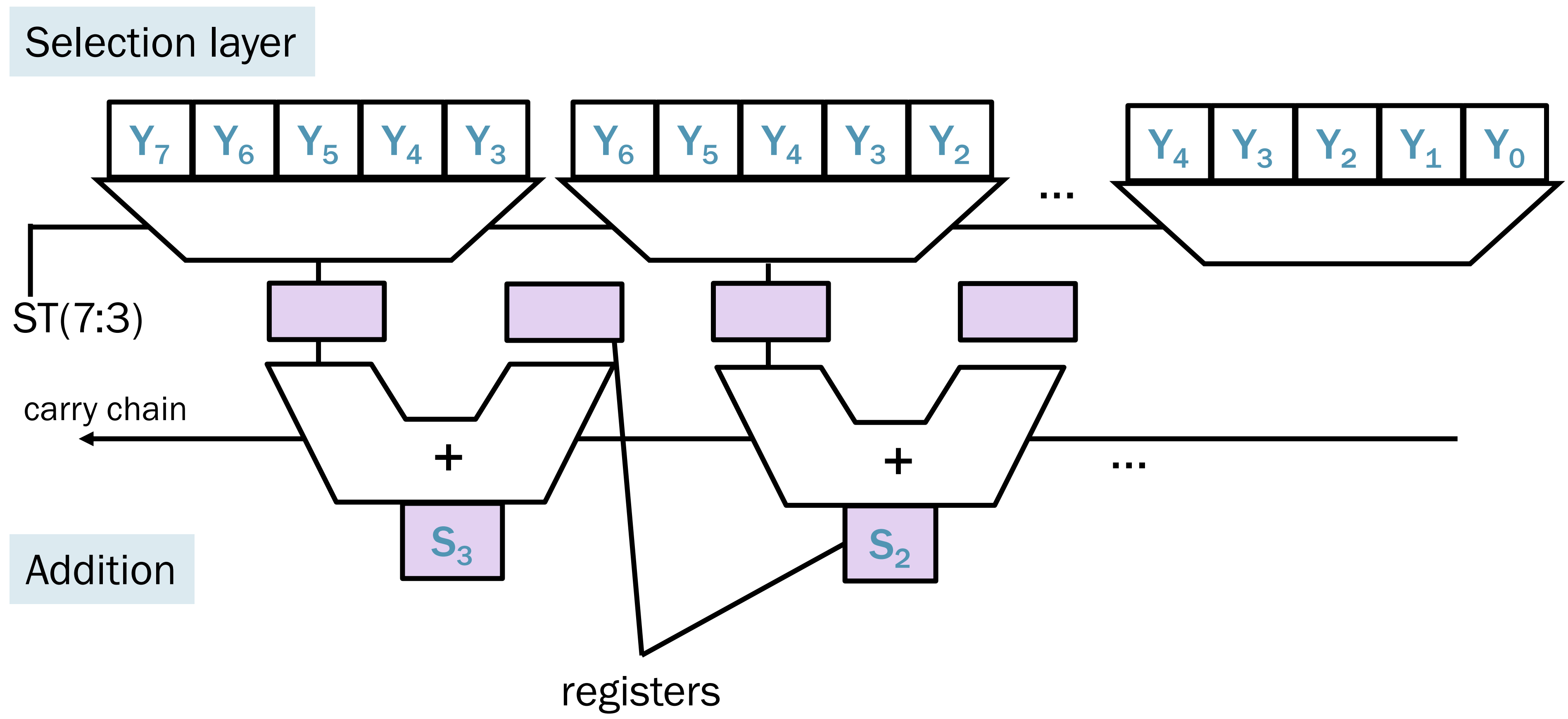
Experimental Results: on FPGA

- Modelled in VHDL, simulated & verified with ModelSim
- Tools: Quartus Prime Lite 18.1
  Target: Cyclone V
- Application: Fibonacci series generator

Design	Comb. ALUTs
Accurate 32-bit adder	**32**
Approximate adder 32-bit AIF	**183**
Selection layer	80
Decompression layer	48
Addition layer	20
Alignment layer	24
ST generation	11

Timing Analysis:
Fmax: 99.56 MHz

Pipelining the AIF Adder


Selection layer
Y_7 Y_6 Y_5 Y_4 Y_3
Y_6 Y_5 Y_4 Y_3 Y_2
…
Y_4 Y_3 Y_2 Y_1 Y_0
ST(7:3)
carry chain
+
+
…
S_3
S_2
Addition
registers

Results: Pipelined AIF Adder

- FPGA is well optimized with carry chains and rich in registers.
- 3-stage pipeline: after selection and addition layers
- This improves throughout but the results show some penalty in latency.

Design	Comb. ALUTs
Accurate 32-bit adder	**32**
3-stage Pipelined Approximate adder 32-bit AIF	**168**
Selection layer	61
Decompression layer	48
Addition layer	20
Alignment layer	25
ST generation	14

Timing Analysis:
Fmax: 167.20 MHz

Dedicated Logic Registers: 66

Conclusion

- Despite optimistic results presented in the paper (e.g. 50% normalized power consumptions), AIF suffers from significant overheads that hinders its practical usages.
- The decompression, selection and alignment of data blocks are intrinsic to the format. This definitely slows down the whole AIF addition, despite fewer bits to add.
- Pipelining can help, under the assumption that AIF additions are carried out in batches.

Future (Unfinished?) work

- Customization based on pc
- Extension to subtraction/signed adder
- Extension to multiplier
- Performance and power analysis based on standard cell synthesis and implementation

Any questions?

Matthew Tang
matthew.tang@qmul.ac.uk