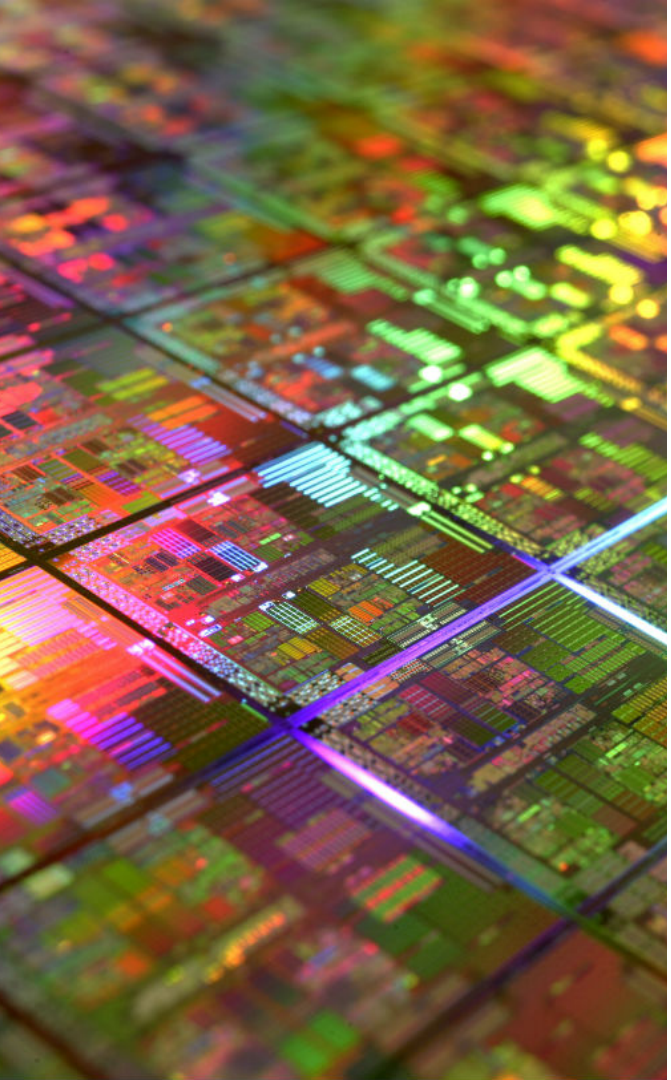


CC-Spin: A Micro-architecture design for scalable control of Spin-Qubit Quantum Processor

Amitabh Yadav
Research Associate, Physics Division, LBNL
Quantum Computer Architecture Lab, QuTech

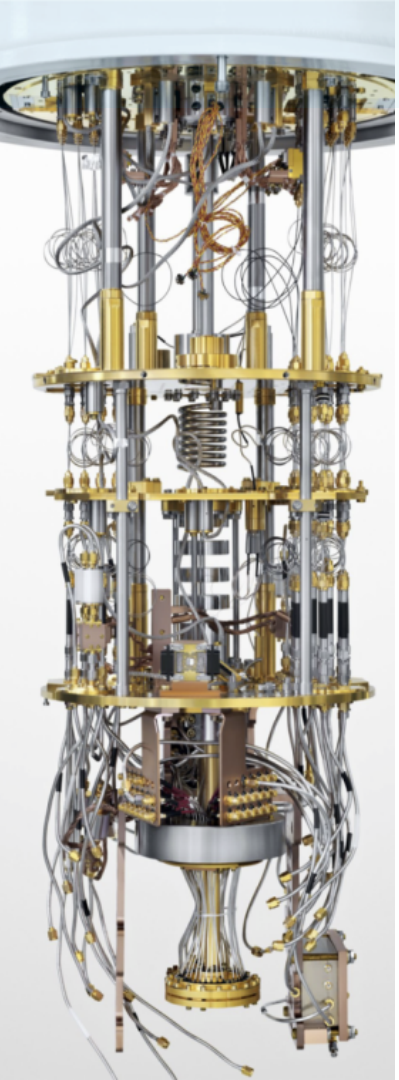
Presentation based on
Thesis Ref. No. Q&CE-QCA-MS-2019-16

April 2, 2020

Supervisors:

Dr. Nader Khammassi
Intel Quantum Computing Lab, Hillsboro

Dr. Koen Bertels
Professor, QCA Lab/QuTech, TU Delft



Outline

Motivation

- CMOS-era
- Beyond-CMOS & Second Quantum Revolution

Requirements Study
Understanding the Hardware

- Spin Qubit Control and Requirements
- Superconducting Qubits Control and Requirements

CC-Spin Micro-architecture

- Microarchitecture: The system view
- Instruction Format
- Quantum Pipeline
- Microcode Unit
- Microarchitecture for NISQ-era Algorithms

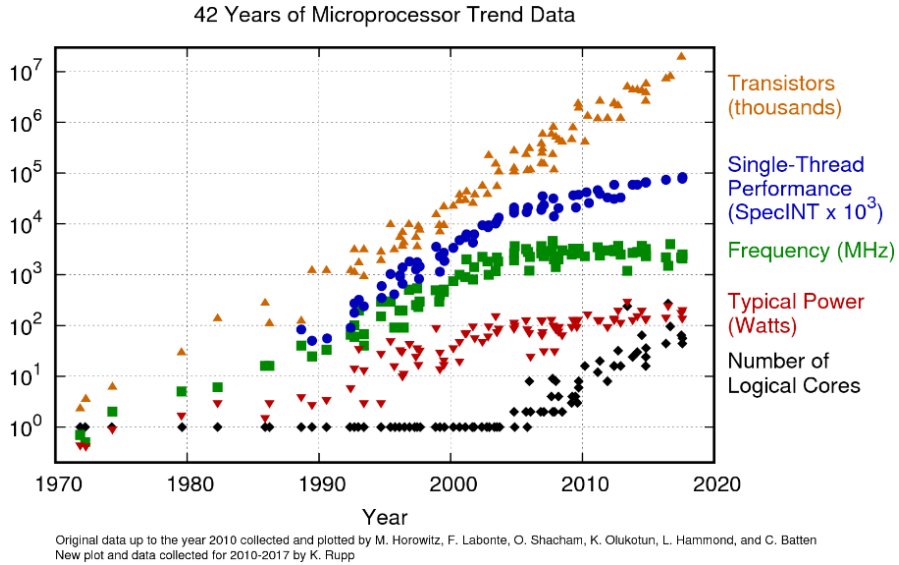
Testing and Results

- Quantum Pipeline
- FPGA controlled DAC-based AWG
- DDS-based AWG architecture

Conclusion

- Conclusion & Future Works

Beyond CMOS-era



Jim Harris ✓
@JimHarris

FUNNY! & TRUE! The Number Of People Predicting The Death Of Moore's Law Doubles Every 18 Months #SAPPHIRENOW #tech #Intel #computing #ASUG17

TU Delft

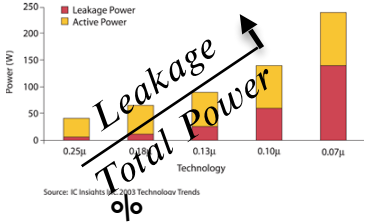
Beyond CMOS-era

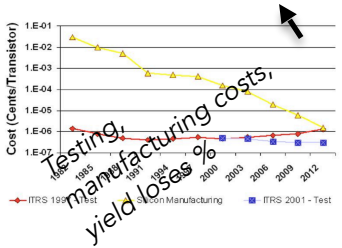
Walls : CMOS device technology

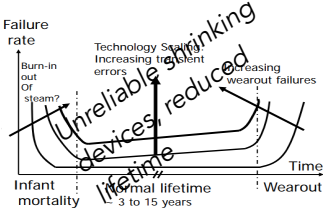
- Leakage wall
- Cost wall
- Reliability wall

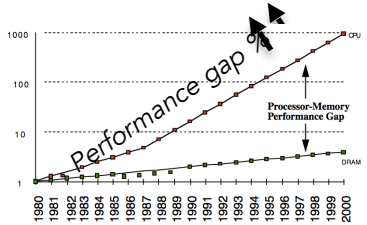
Walls: Von-Neumann architectures

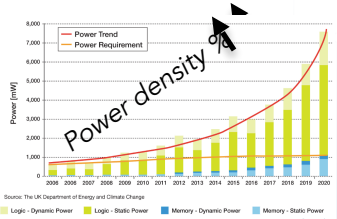
- Memory wall
- Power wall
- Instruction Level Programming wall

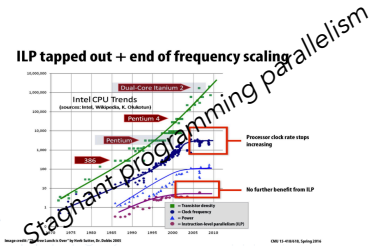












Quantum Computing

Technologies

- Superconducting Qubit
- Spin-Qubit Quantum Dot
- NV centers in Diamond
- NMR, Ion-Traps etc…

TU Delft

Second Quantum Revolution

Quantum Bits!

But how to control them?

Second Quantum Revolution

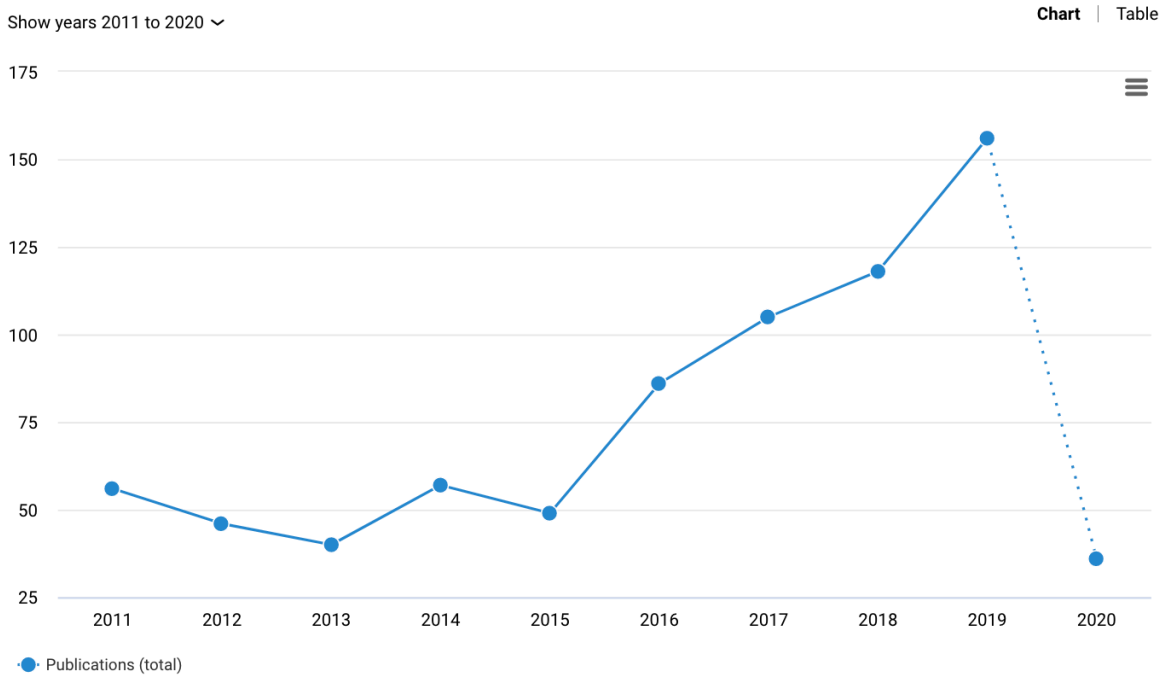
Google 72-qubit chip set-up. Image Source: Internet.

TU Delft

Second Quantum Revolution

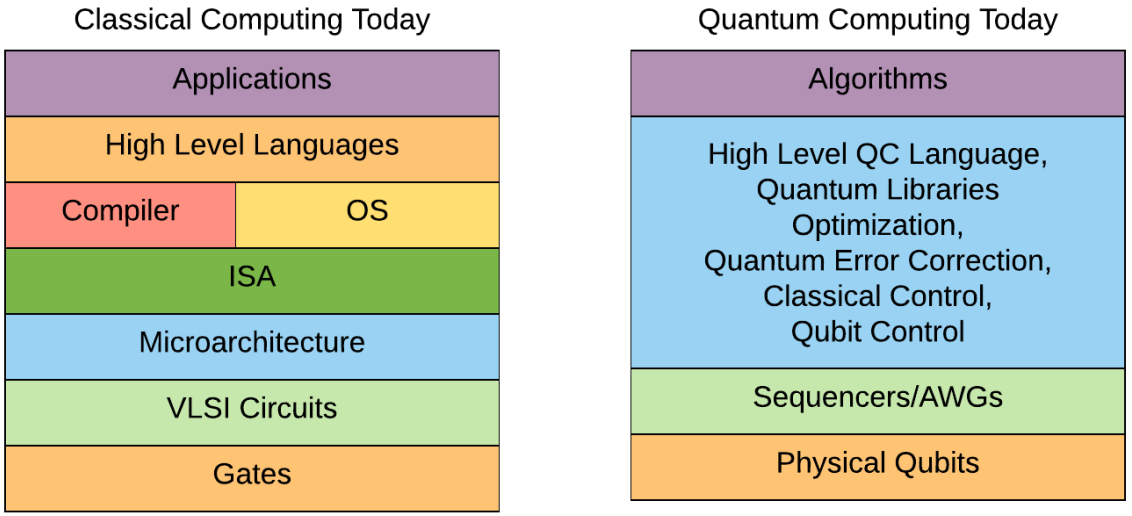
Trends in quantum control technology

Largely unexplored territory!

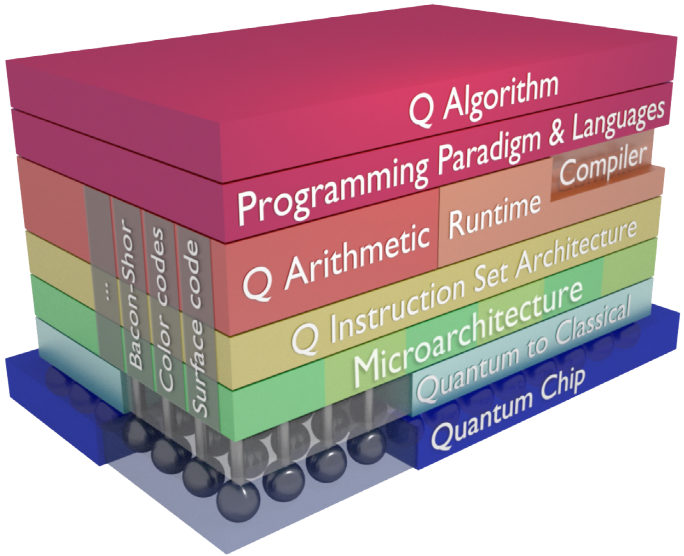


Source: "Quantum Computer Architecture", Publications Report, Dimensions

QCA Full-Stack Architecture



QCA Full-Stack Architecture



Grover's Search, Deutsch-Jozsa, Shor's etc.

OpenQL

↓ **OpenQL Compiler**

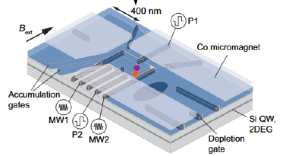
cQASM

eQASM — Hardware

Microarchitecture

Analog Waveform generation

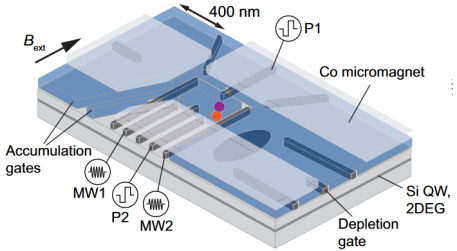
↓ **Quantum-to-Classical Interface**

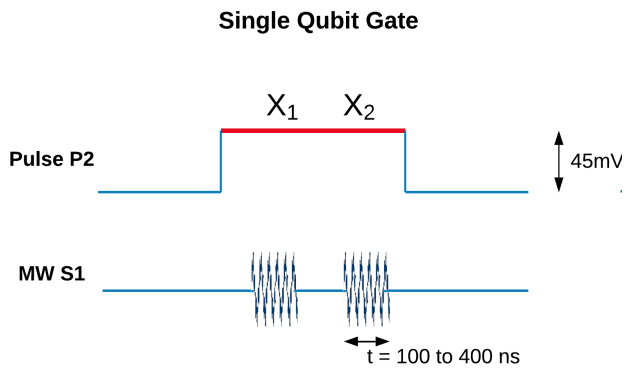


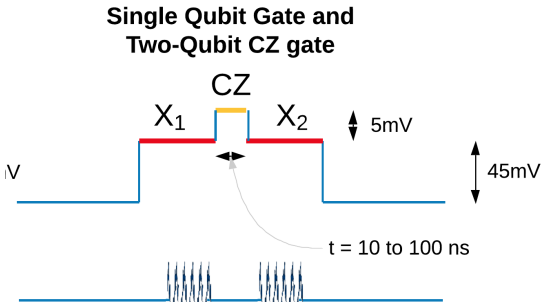
TU Delft

Spin Qubit Control Requirements

- **Initialization and Measurement:** Controlling the movement (of electron in/ out of Quantum Dot) with **tiny AC pulses (40 – 150 ns duration.)**
- **Single Qubit Gates:** Done by energised microwave photons.
  Signal: Short Microwave pulses (18 – 20GHz, 40-150ns duration).
- **Two qubit Gates:** pulsing the barrier gate causes wave functions overlap.
  Signal: DC pulse (10ns precision)





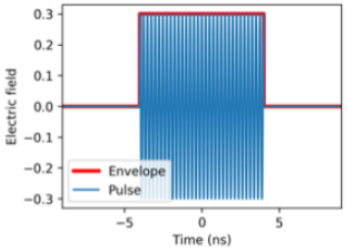


J. P. van Dijk, E. Charbon, and F. Sebastiano, *The electronic interface for quantum processors,* Microprocessors and Microsystems **66**, 90 (2019)

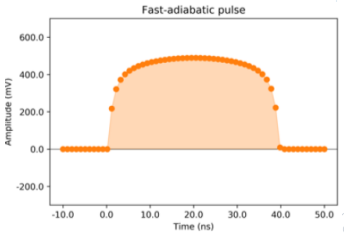
T. Watson, S. Philips, E. Kawakami, *et al.*, *A programmable two-qubit quantum processor in silicon,* Nature **555**, 633 (2018).

Superconducting Qubits Control Requirements

- **Measurement:** **Microwave pulse (7-8 GHz) for 300ns – 2μs duration** via readout resonator.
- **Single Qubit Gates:** via Microwave drive line or via readout resonator**.**
  Signal: Microwave Signal 4-10GHz, 20ns duration.
- **Two qubit Gates (CZ):** Applying current to Flux-bias line.
  Signal: Fast Adiabatic Pulses (pulse shaping), 40ns duration.



Single Qubit Gate Pulse

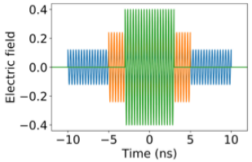


Two Qubit Gate Pulse

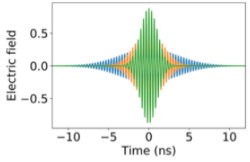
Image source:
J. Martinis group

Comparison

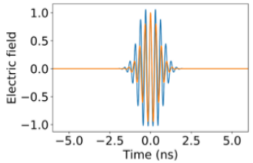
Superconducting Qubit Control



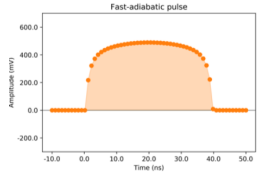
Short and Fast Pulses



Gaussian Envelop Pulses



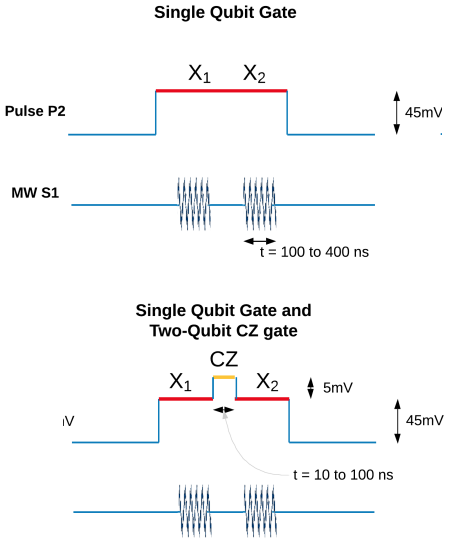
DRAG Pulses
(Derivative Removal by Adiabatic Gate)
Requires Pulse shaping, <10ns duration, >0.999 fidelity



Two Qubit Gate Pulse

Image source:
J. Martinis group

Spin Qubit Control



Data source: QuTech

SC/Spin Qubit Control:

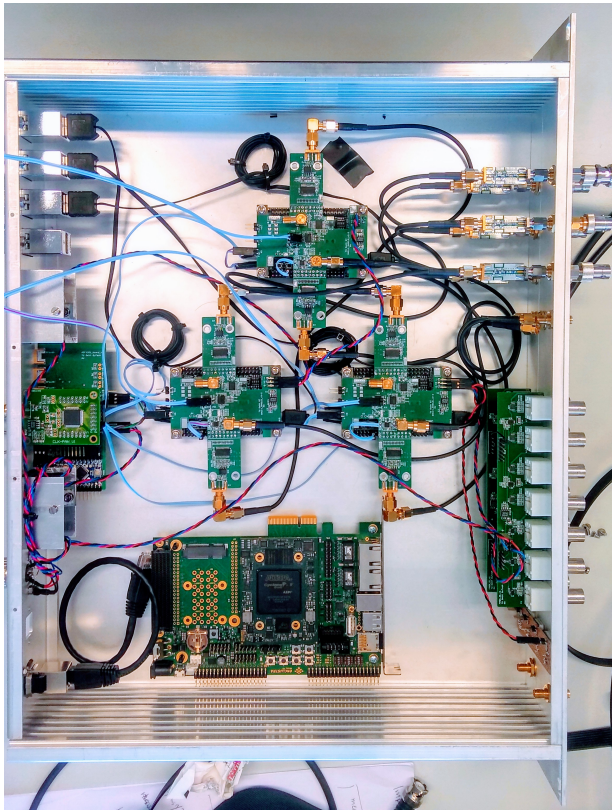
- Pulse Signal (AC/DC),
- Microwave pulses (3-20GHz, mainly Sinusoids)
  - fast-addressable parameterized waveforms
  - pulse shaping capabilities – Gaussian Envelopes, DRAG pulses etc.
  - IQ and DC output channels

Predicted: 40+ connections for 5 qubit chip

- Scaling would be linear or worse
- need more output channels
- modular architecture: 100s (to 1000s) of qubit control

TU Delft

CC-Spin Micro-architecture

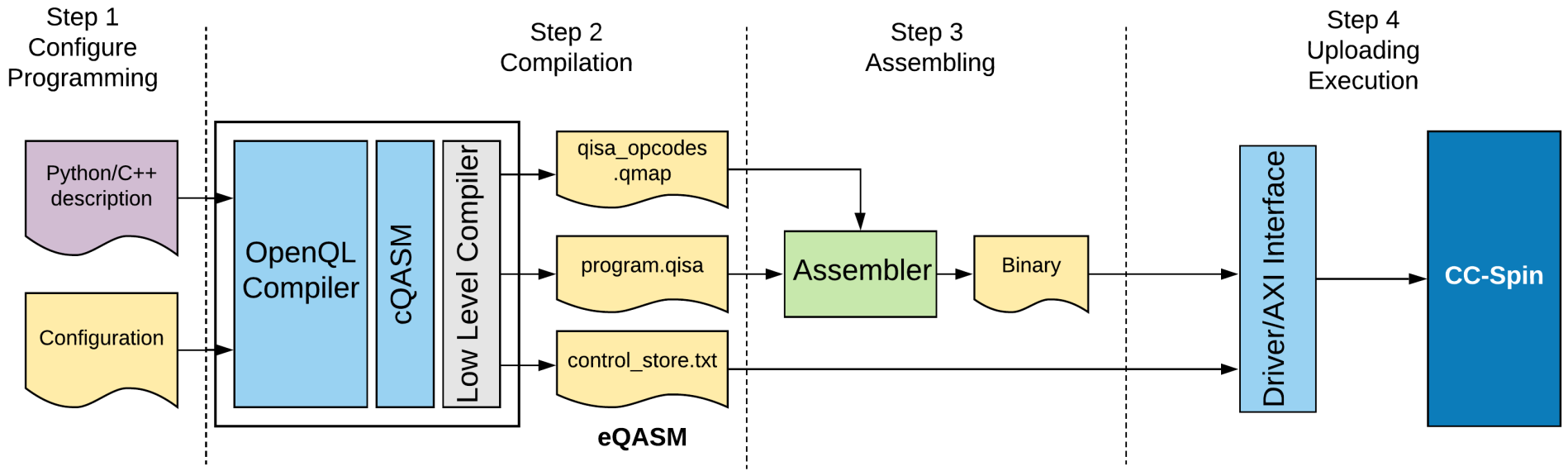


QCA/QuTech's **CBoxV3** currently under modification for CC-Spin Microarchitecture

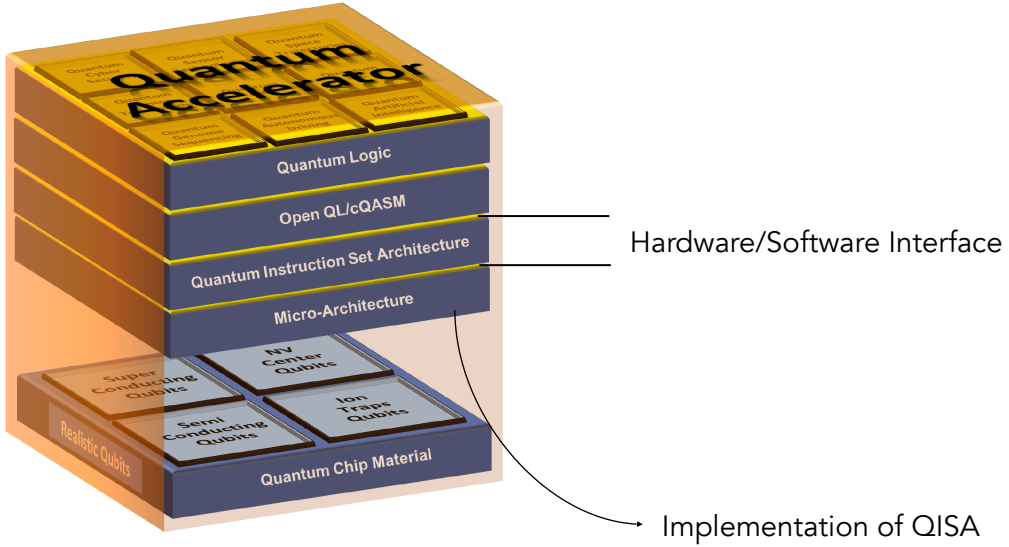
Microarchitecture

- Abstraction Layer between the HL programming & low-level control/Digital & Analog using a QISA.
- Independent of the technology implementation.
- Generates real time high-speed signals (DC, Sinusoidal or Arbitrary) with precise timing control.
- Implements fast readouts, and readout-based control (feed-forward control) of the quantum chip.
- Implements the classical control logic for quantum error decoding and correction.

Q/C Instruction Execution Flow



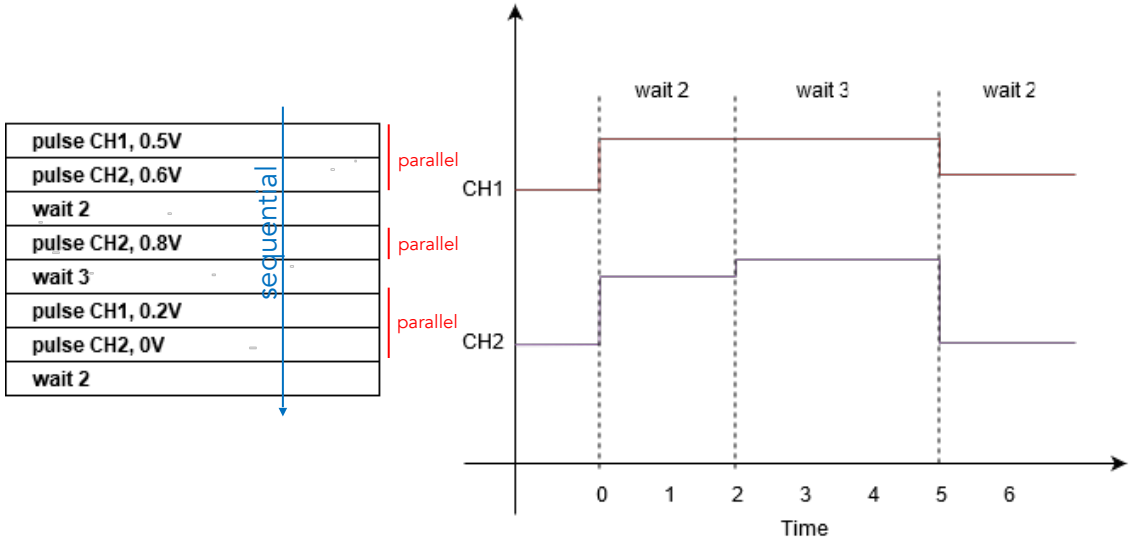
Quantum Instruction Set Architecture (QISA)



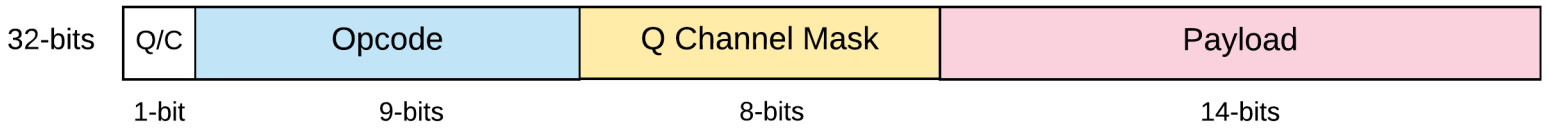
QISA

- Classical and quantum instructions, including run-time feedback execution
- Timing information of quantum operations.
- No Qubit technology-dependent constraints.
- Densely encoded for high instruction issue rate.
- Flexible to accommodate new sets of quantum operations and support complex waveform generation.

Instruction Format



A General Instruction format:



QISA

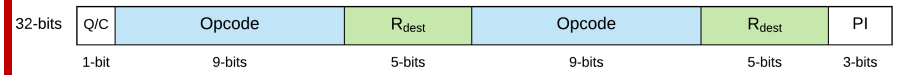
Ver. 1

Wait Instruction:

ISA for Parameterized Waveform Generation:

Wait Instruction with Registers:

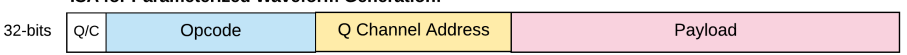
Stored Waveform Quantum Instruction Execution using Registers:



Ver. 2

Wait Instruction:

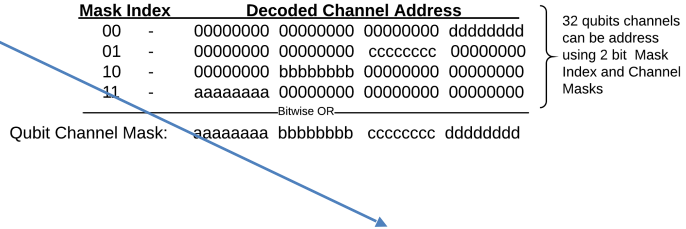
ISA for Parameterized Waveform Generation:

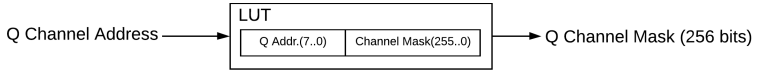


Wait Instruction with Registers:

Stored Waveform Quantum Instruction Execution using Registers:

SIMQ

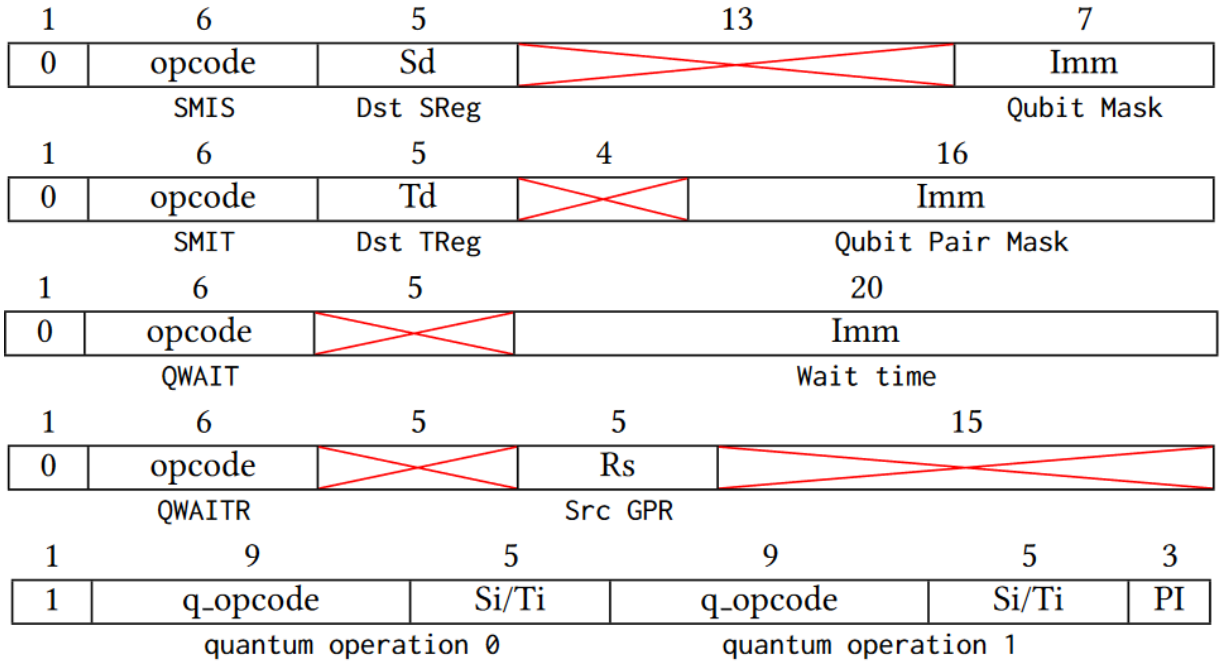




Single Instruction Multiple Qubits

eQASM Format for CTPG

Qubit Registers for Single and Two-qubit parallel operations



Quantum Instruction

Wait Instructions in terms of number of cycles/32-bit register value

X.Fu et.al. eQASM (2018)

Example of QISA definition: eQASM Format for CTPG

Classical Instructions

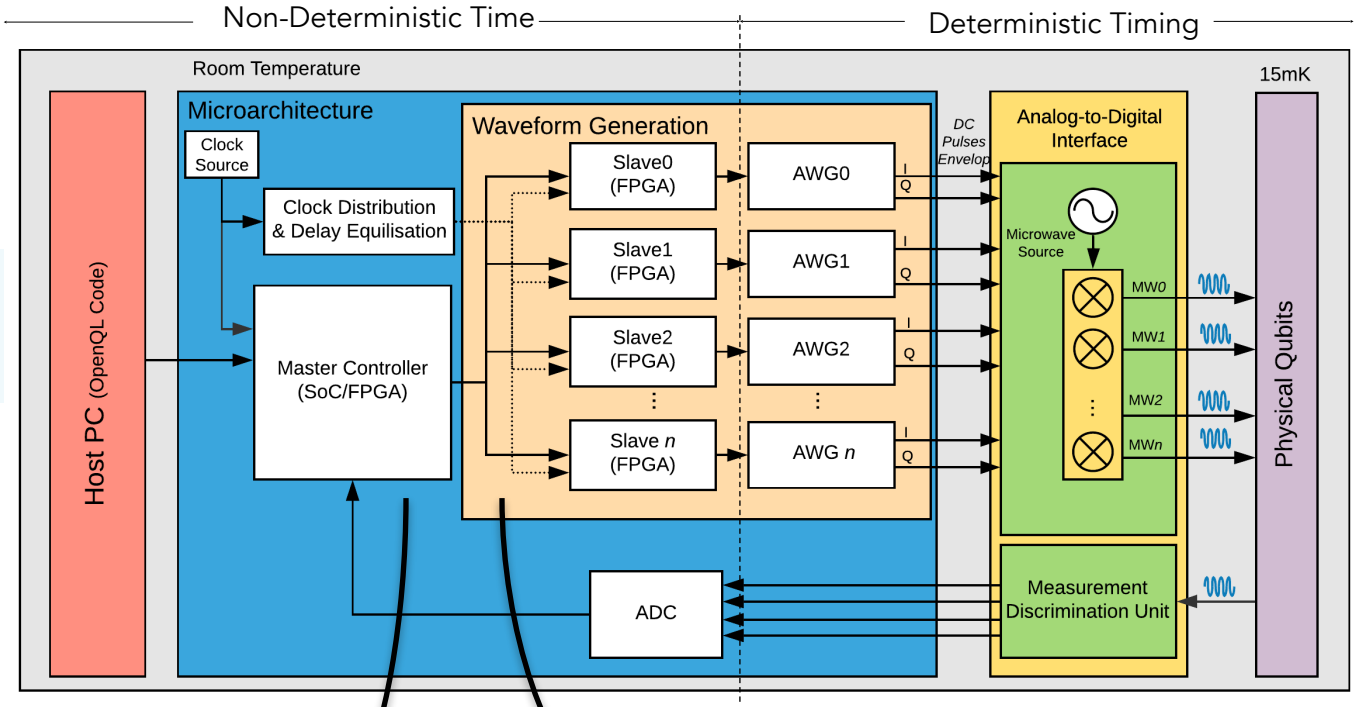
Type	Syntax	Description
Control	`CMP Rs, Rt`	CoMPare GPR Rs and Rt and store the result into the comparison flags.
	`BR <Comp. Flag>, Offset`	(BRanch) Jump to PC + Offset if the specified comparison flag is '1'.
Data Transfer	`FBR <Comp. Flag>, Rd`	(Fetch Branch Register) Fetch the specified comparison flag into GPR Rd.
	`LDI Rd, Imm`	(LoaD Immediate) Rd = sign_ext(Imm[19..0], 32).
	`LDUI Rd, Imm, Rs`	(LoaD Unsigned Immediate) Rd = Imm[14..0]::Rs[16..0].
	`LD Rd, Rt(Imm)`	(LoaD from memory) Load data from memory address Rt + Imm into GPR Rd.
	`ST Rs, Rt(Imm)`	(STore to memory) Store the value of GPR Rs in memory address Rt + Imm.
	`FMR Rd, Qi`	(Fetch Measurement Result) Fetch the result of the last measurement instruction on qubit i into GPR Rd.
Logical	`AND/OR/XOR Rd, Rs, Rt` `NOT Rd, Rt`	Logical and, or, exclusive or, not.
Arithmetic	`ADD/SUB Rd, Rs, Rt`	Addition and subtraction.
Waiting	`QWAIT Imm` `QWAITR Rs`	(Quantum WAIT Immediate/Register) Specify a timing point by waiting for the number of cycles indicated by the immediate value Imm or the value of GPR Rs.
Target Specify	`SMIS Sd, <Qubit List>` `SMIT Td, <Qubit Pair List>`	(Set Mask Immediate for Single-/Two-qubit operations) Update the single- (two-)qubit operation target register Sd (Td).
Q. Bundle	`[PI,] Q_Op [| Q_Op]*`	Applying operations on qubits after waiting for a small number of cycles indicated by PI.

Quantum Instructions

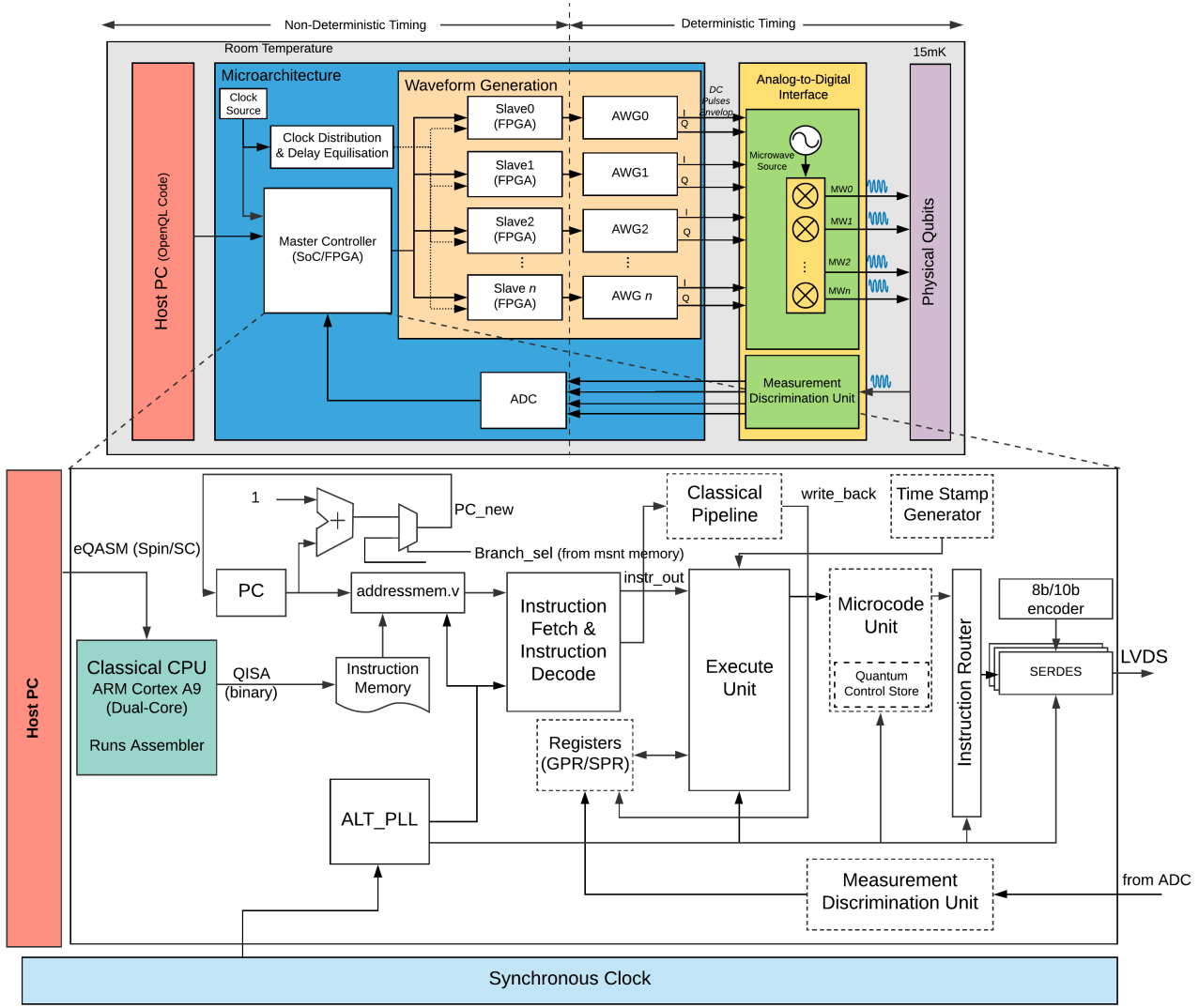
TU Delft

X.Fu et.al. eQASM (2018)

Micro-architecture
The system view

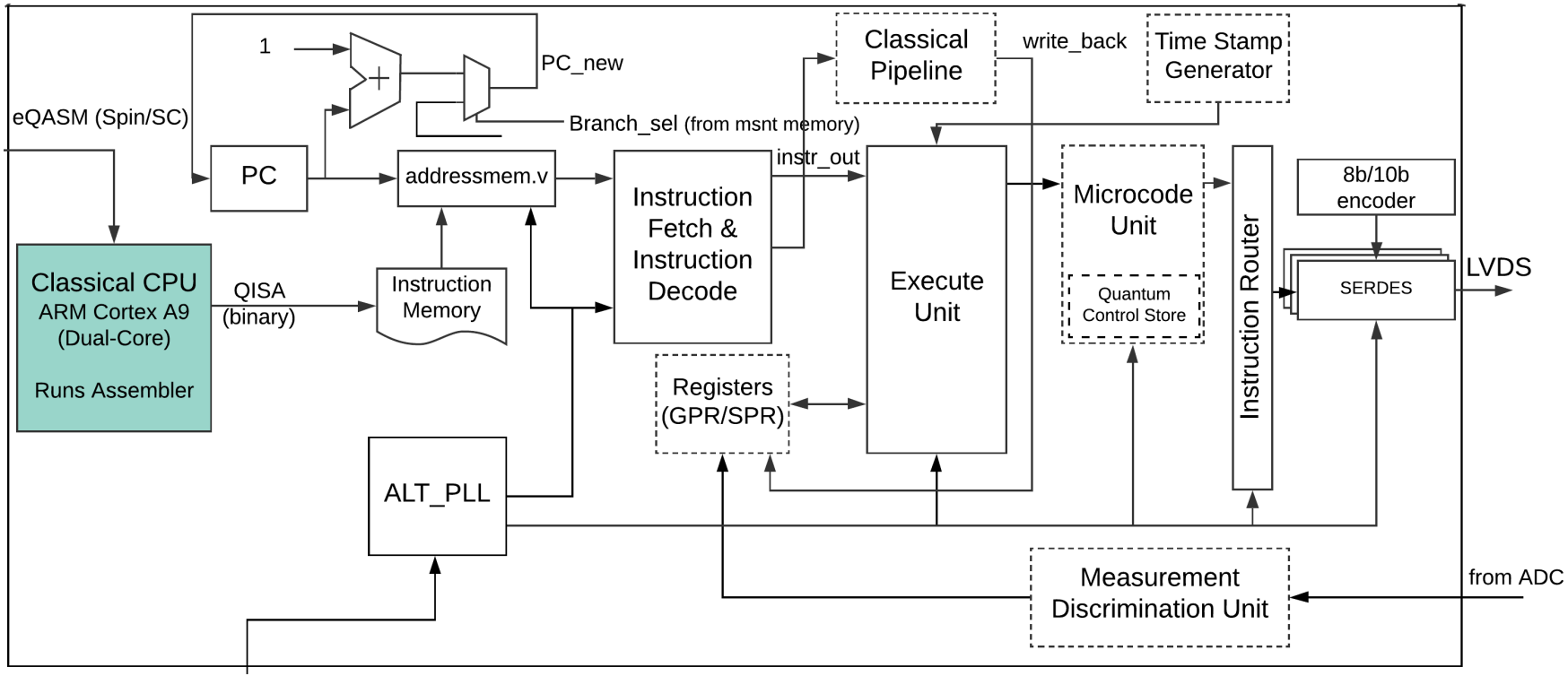


Micro-architecture
The system view



TU Delft

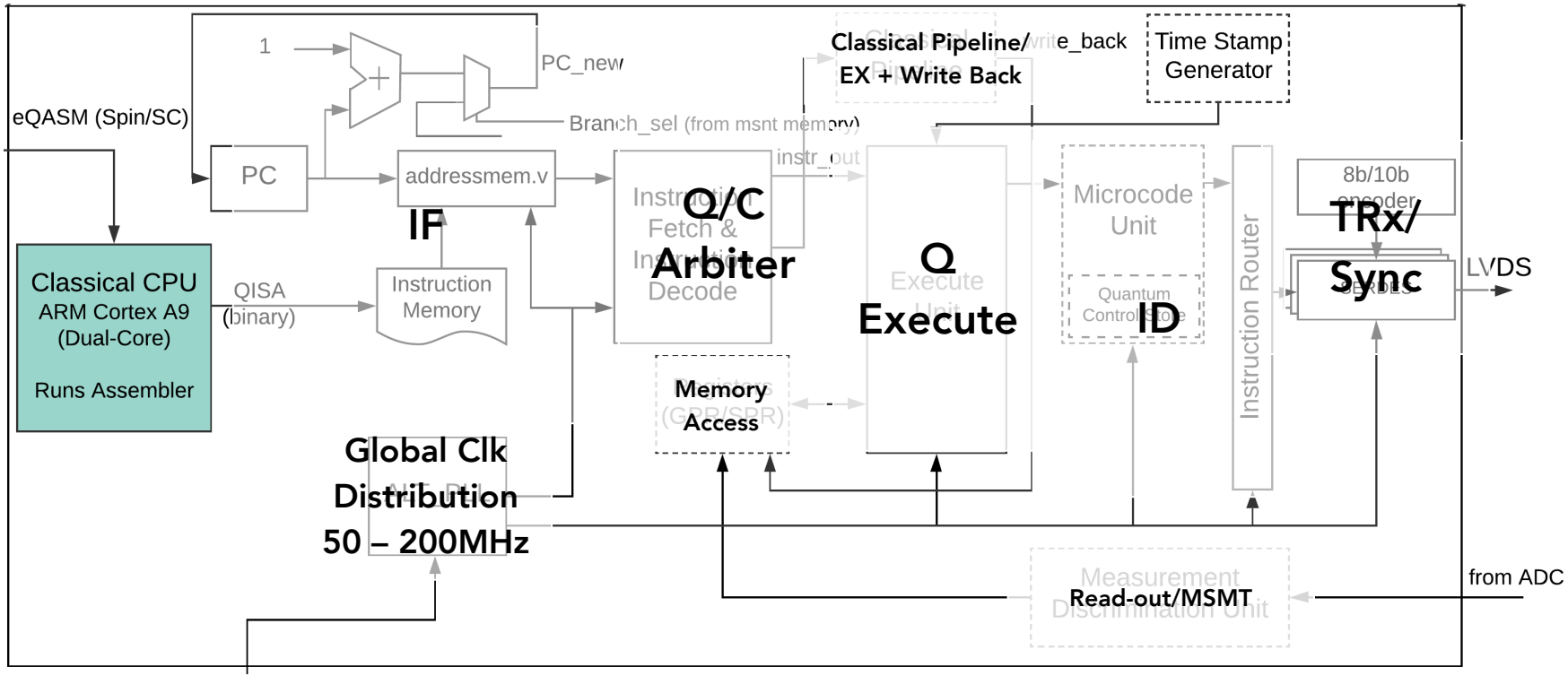
CC-Spin Quantum Pipeline



Note:
Implementation here is that of a Hardwired Decode Unit.
Horizontal Microcode Unit implementation is beyond the scope of the thesis.

Not included in practical implementation

CC-Spin Quantum Pipeline

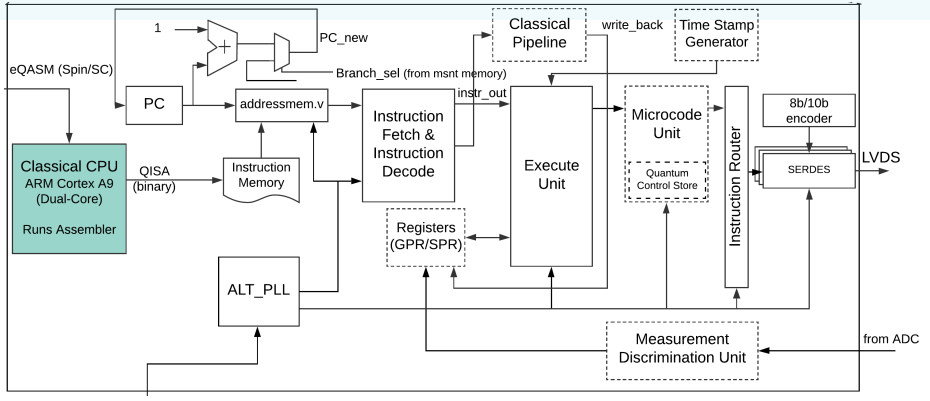


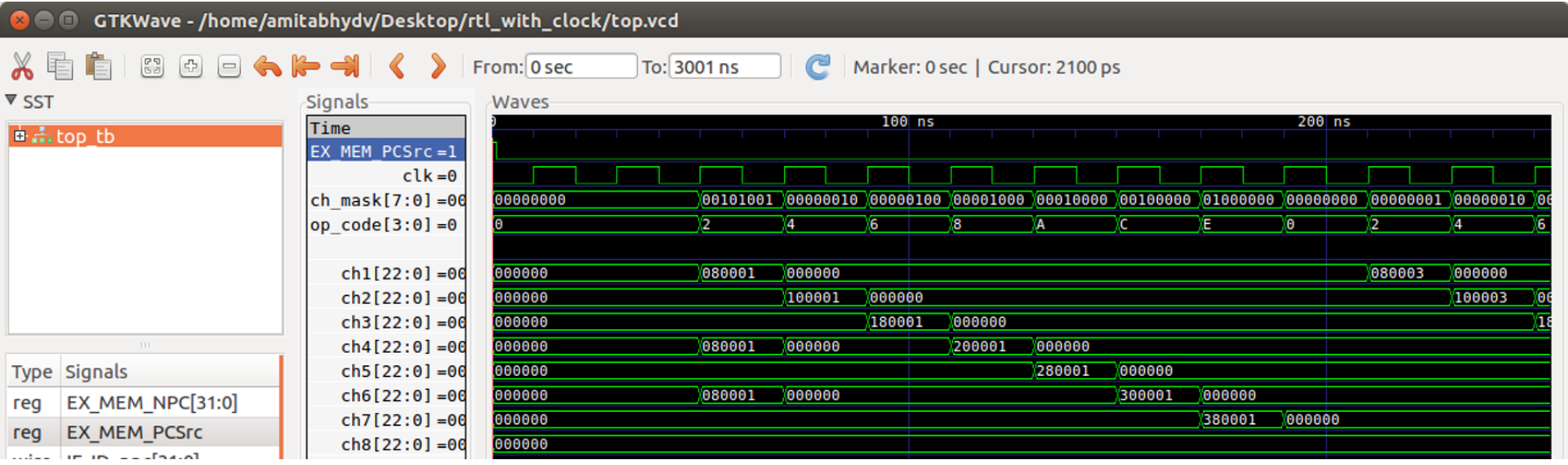
Note:
Implementation here is that of a Hardwired Decode Unit.
Horizontal Microcode Unit implementation is beyond the scope of the thesis.

Not included in practical implementation

CC-Spin Quantum Pipeline Simulation (with General Instruction Format)

A General Instruction format:





An Example Execution Flow[3]

Source: QCA/QuTech, X.Fu. et.al. 1708.07677v1(

Software

OpenQL (C++/Python)

```
int main(int argc  char ** argv)
{
   // load the hardware configuration of the target platform
   ql::quantum_platform starmon("starmon", "starmon_config.json");

   // we create quantum program (we specify the compiler backend)
   ql::quantum_program prog("prog",2,starmon);

   // create a new kernel
   ql::str_t name = "my_kernel";
   ql::quantum_kernel kernel(name);
   kernel.prepz(0);      // prepare q0 in zero state
   kernel.hadamard(0);   // H q0
   kernel.x(1);          // X q0
```

H/W Config. File (.json)

```
{
   "eqasm_compiler" : "qumis_compiler",

   "hardware_settings": {
         "qubit_number": 2,
         "cycle_time" : 5,
         "mw_mw_buffer": 0,
         "mw_flux_buffer": 0,
         …
   },
   "instructions": {
      "rx180 q1" : {
         "duration": 40,
         "latency": 20,
         "qubits": ["q1"],
         "matrix" : [ [0.0,0.0], [1.0,0.0],
                      [1.0,0.0], [0.0,0.0] ],
         "disable_optimization": false,
         "type" : "mw",
         "qumis_instr": "pulse",
         "qumis_instr_kw": {
            "codeword": 1,
            "awg_nr": 2
         }
      },
```

cQASM

```
# this file has been automatically
# generated by the OpenQL compiler

qubits 2

.kernel_1
   { prepz q0 | prepz q1 }
   rx180 q0
   ry90  q0
   cz    q0,q1
   ry180 q0
   measure q0
   measure q1

.kernel_2
   { prepz q0 | prepz q1 }
   ry180 q1
   cz    q1,q0
   ry180 q0
   measure q0
   measure q1
```

eQASM

```
# this file has been automatically
# generated by the OpenQL compiler

wait 1
mov r14, 0
start:
   pulse 0000, 0010, 0000
   pulse 0000, 0000, 0001
   wait 20
   trigger 0011110, 8
   wait 1
   trigger 0000001, 1
   wait 7
   pulse 0011, 0000, 0000
   wait 12
   trigger 1000000, 16
   wait 16
   trigger 0110110, 8
```

Hardware/Software Interfac

Binary ISA

```
1  00000000000000000000000000000001
2  00000000000000000000000000000010
3  00000000000000000000000000000011
4  00000000000000000000000000000100
5  00000000000000000000000000000101
6  00000000000000000000000000000110
7  00000000000000000000000000000111
8  00000000000000000000000000001000
9  00000000000000000000000000001001
10 00000000000000000000000000001010
```

Hardware

Multi-level instruction decoding

Microcode Instructions

```
# round 0:
   Wait 40000
   Pulse {q0}, I
   Wait 4
   Pulse {q0}, I
   Wait 4
   MPG {q0}, 300
   MD {q0}, r7
# round 1:
   Wait 40000
   Pulse {q0}, X180
   Wait 4
   Pulse {q0}, X180
   Wait 4
   MPG {q0}, 300
   MD {q0}, r7
```

Micro-operations

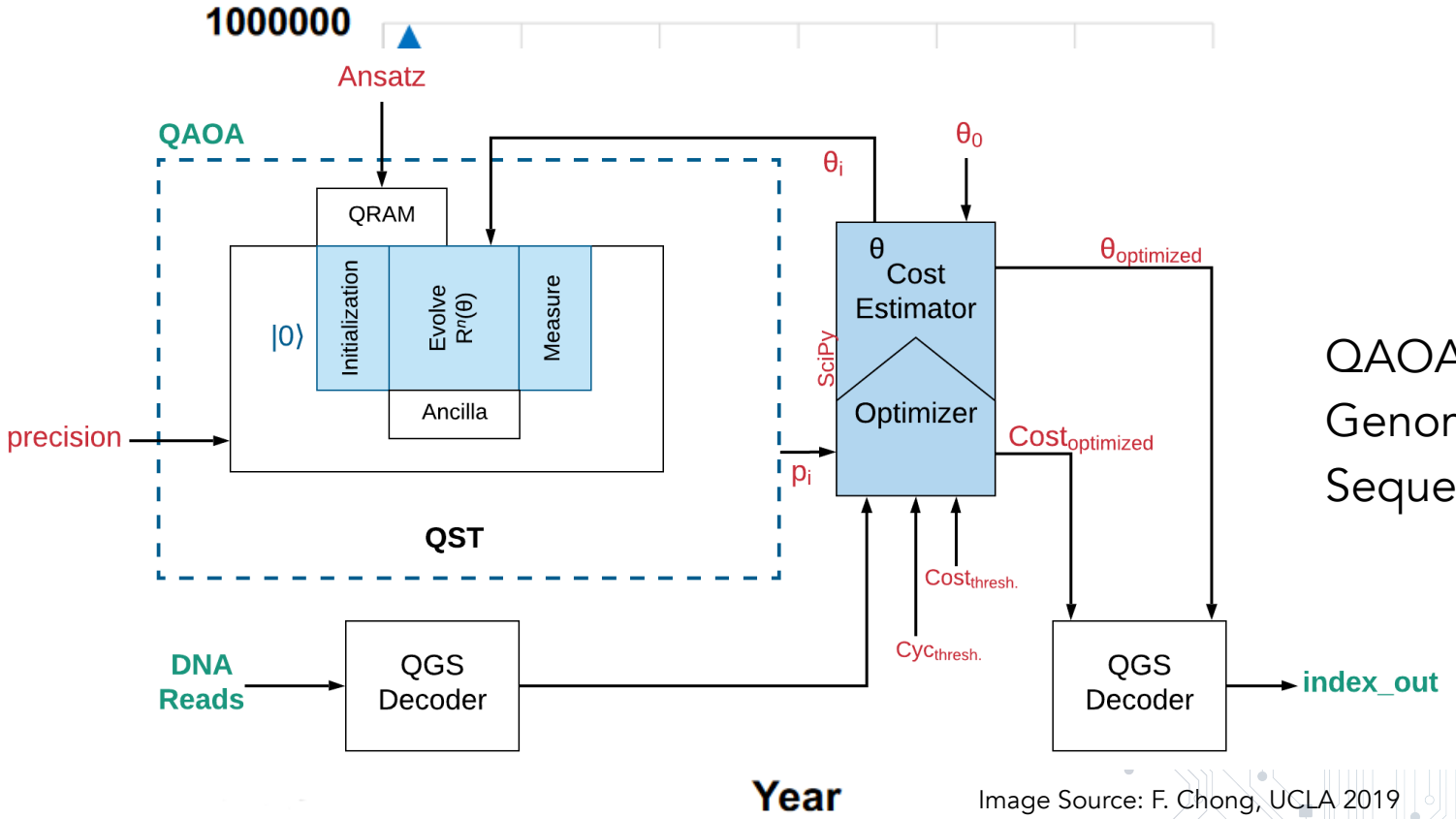
$T_D = 40000$:
  I sent to u-op unit0
$T_D = 40004$:
  I sent to u-op unit0
$T_D = 40008$:
  # MPG&MD bypass this stage
$T_D = 80008$:
  X_π sent to u-op unit0
$T_D = 80012$:
  X_π sent to u-op unit0
$T_D = 80016$:
  # MPG&MD bypass this stage

Codeword Triggers for Stored-waveform AWGs

```
# Δ is the delay of the u-op unit
```

$T_D = 40000 + \Delta$:
  CW 0 sent to CTPG0
$T_D = 40004 + \Delta$:
  CW 0 sent to CTPG0
$T_D = 40008$:
  CW 7 sent to CTPG5 # Msmt
  $MD(r7)$ sent to MDU0
$T_D = 80008 + \Delta$:
  CW 1 sent to CTPG0
$T_D = 80012 + \Delta$:
  CW 1 sent to CTPG0

Micro-architecture for NISQ-era

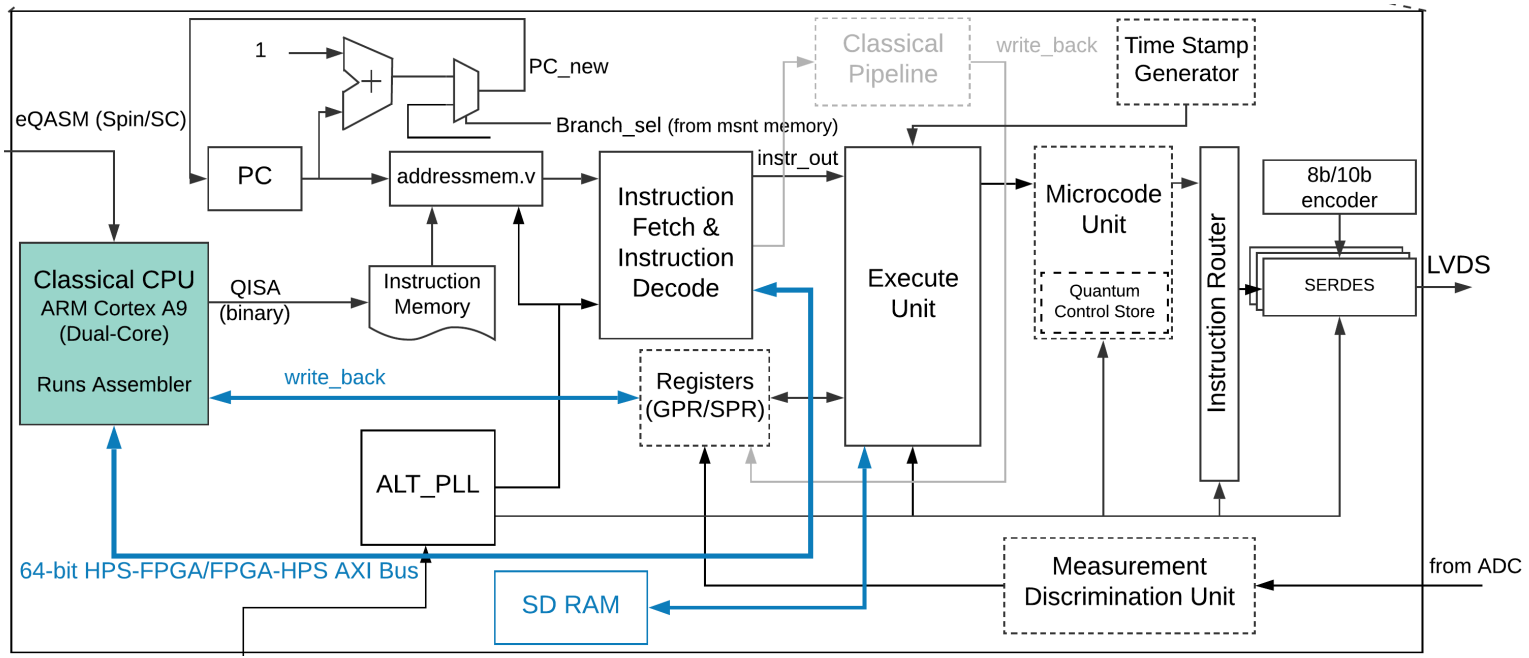


QAOA* for Genome Sequencing

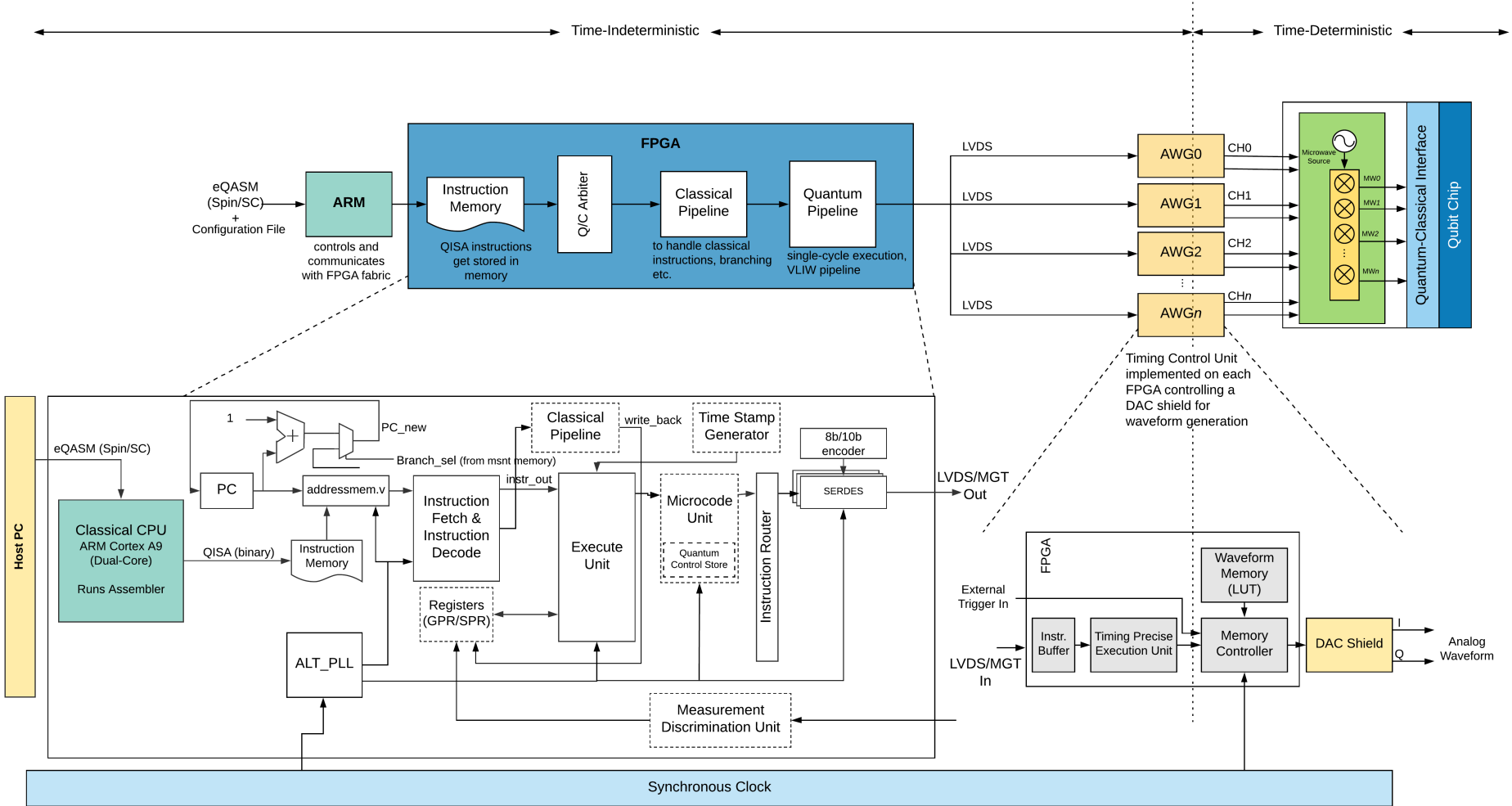
Image Source: F. Chong, UCLA 2019

QAOA: Quantum Approximate Optimization Algorithm
* a type of variational quantum algorithm

Micro-architecture for NISQ-era



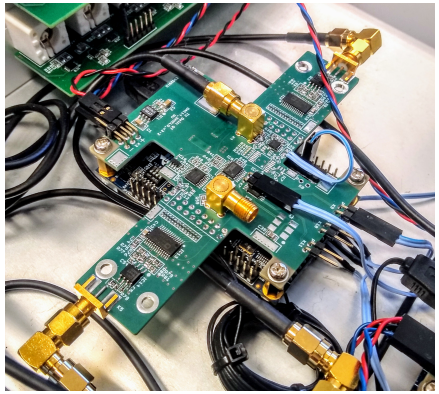
A complete, distributed micro-architecture for Waveform Generation



Waveform Generator Units

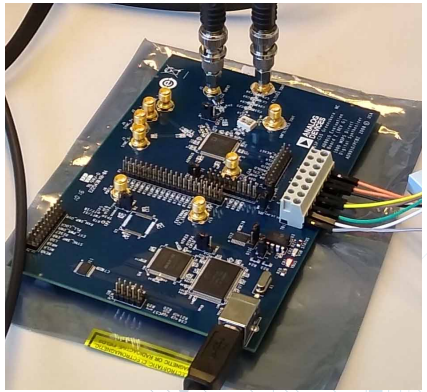
FPGA-controlled DAC-based AWG

for code-word triggered pulse generation



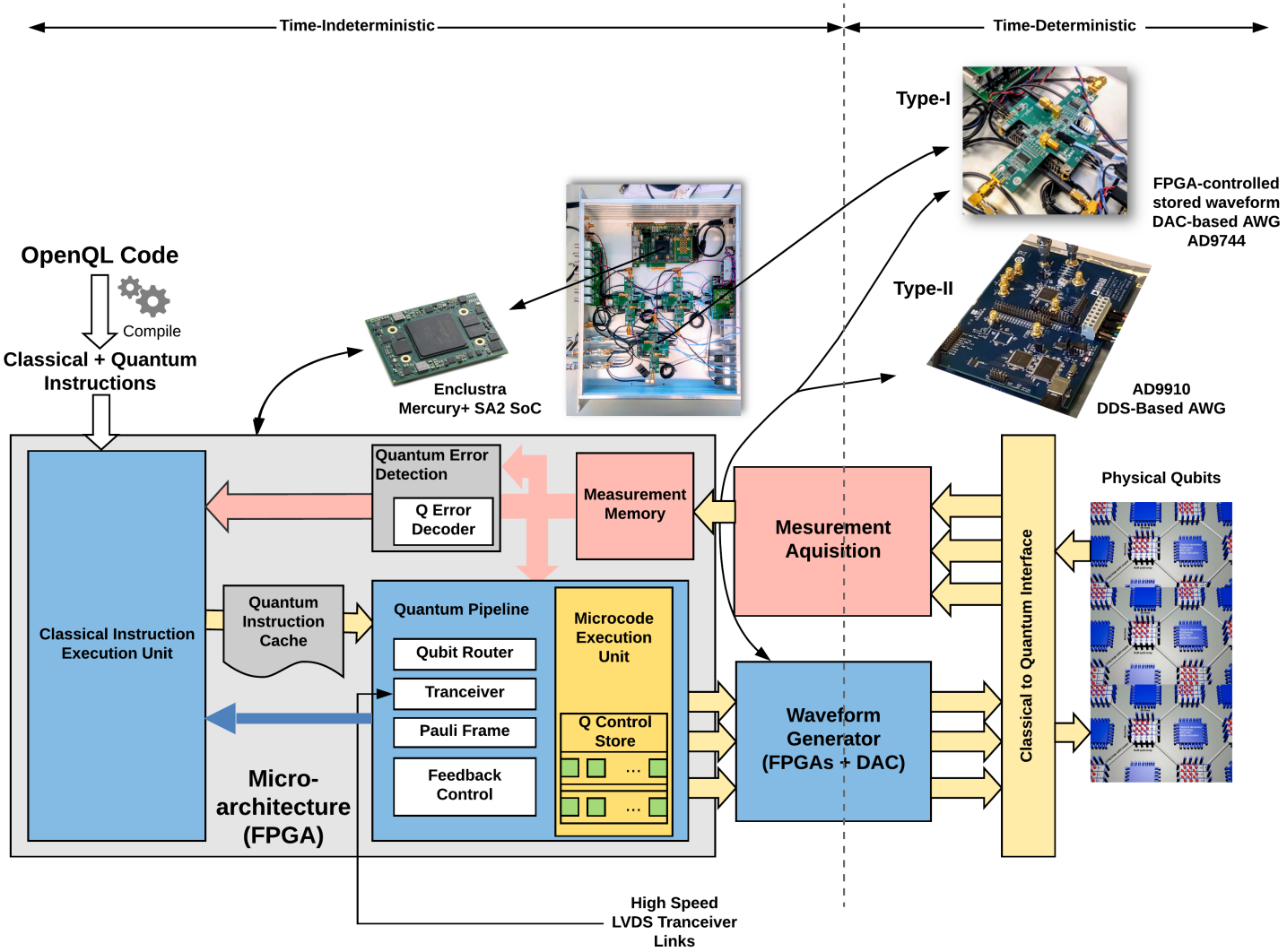
FPGA synchronized DDS-based AWGs

real-time parameterized waveform generation



TU Delft

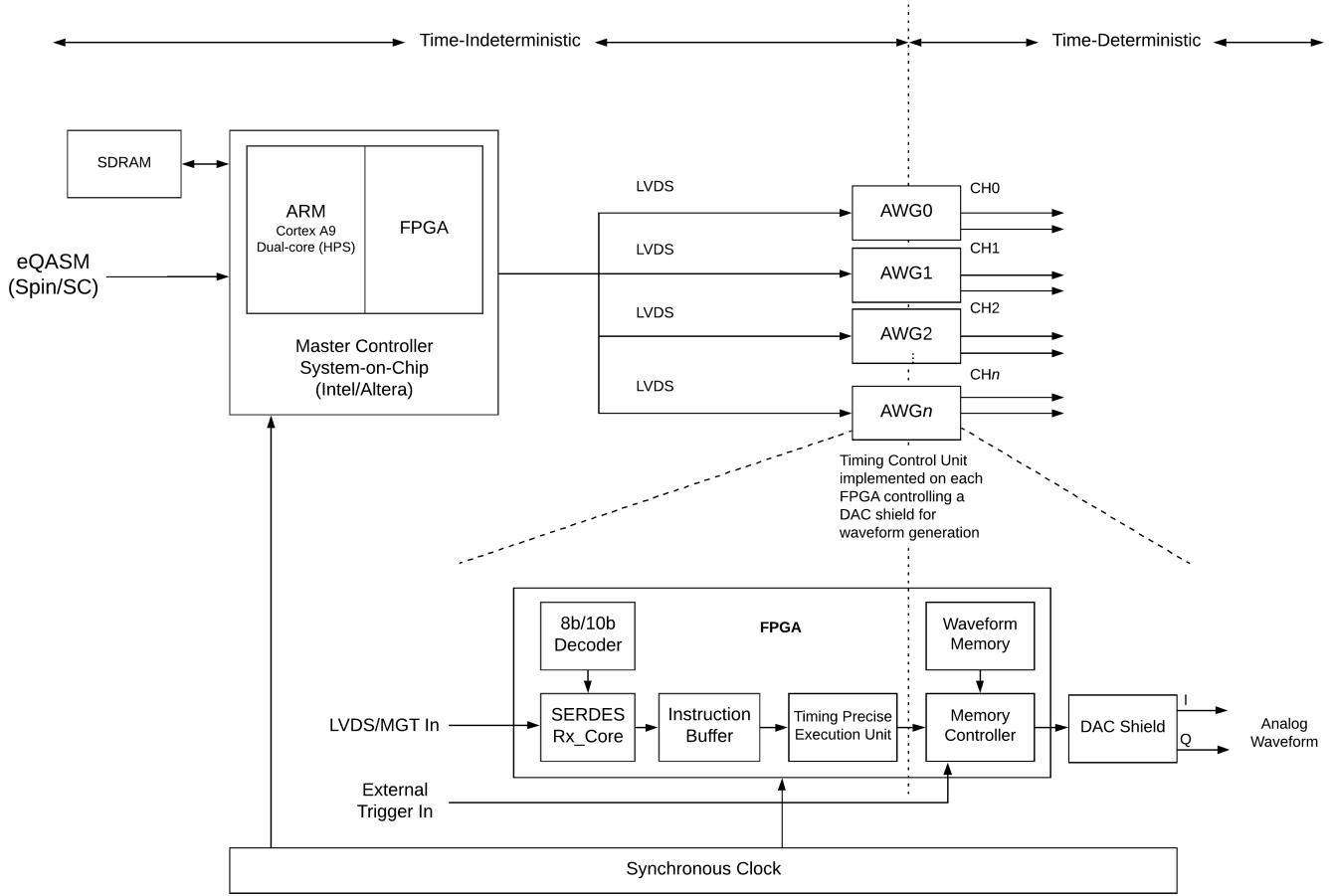
Micro-architecture & Waveform Generation Units



TU Delft

Waveform Generator Units

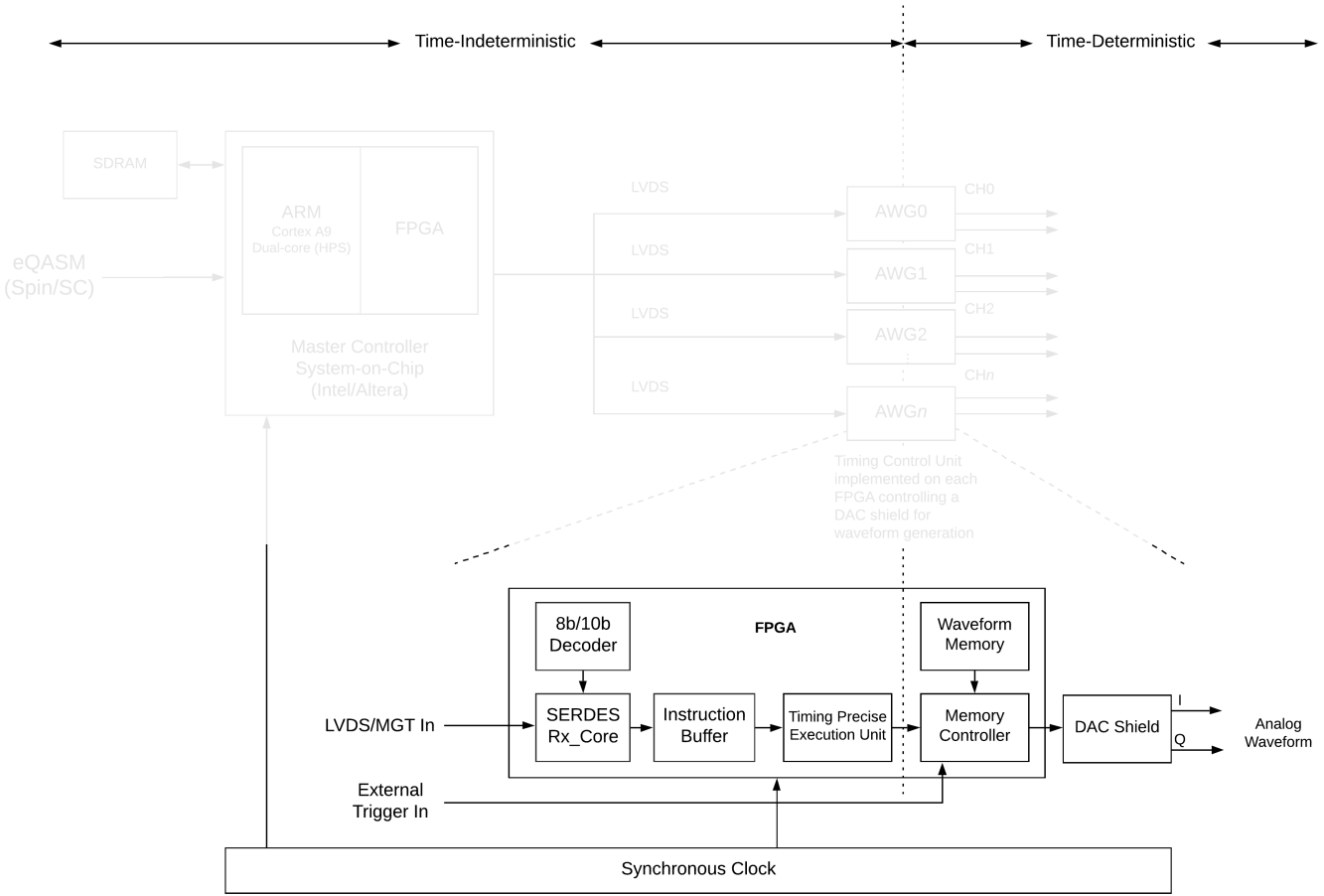
FPGA-controlled
DAC-based AWG



TU Delft

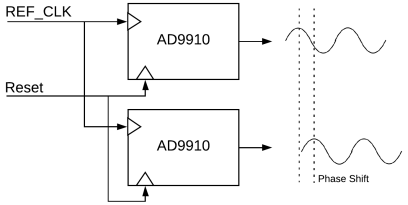
Waveform Generator Units

FPGA-controlled
DAC-based AWG



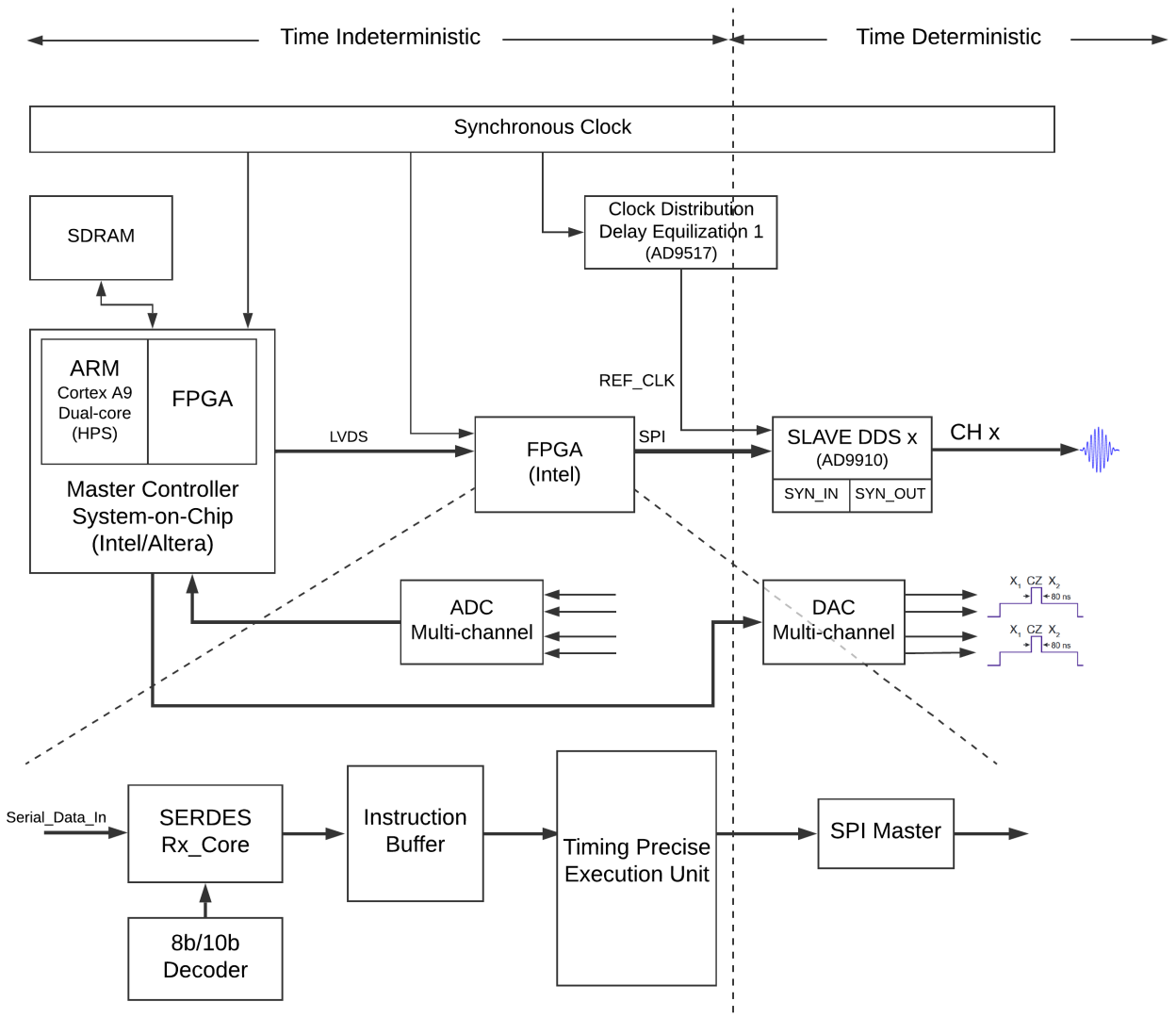
TU Delft

Waveform Generator Units



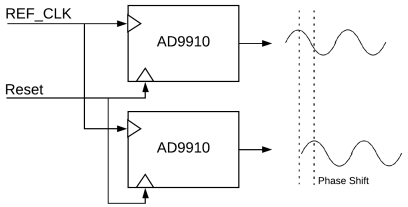
FPGA synchronized DDS-based AWGs

- uses frequency, phase and amplitude data to generate real-time waveform output



TU Delft

Waveform Generator Units



FPGA synchronized DDS-based AWGs

- uses frequency, phase and amplitude data to generate real-time waveform output

Time Indeterministic

Time Deterministic

Synchronous Clock

SDRAM

Clock Distribution
Delay Equilization 1
(AD9517)

ARM
Cortex A9
Dual-core
(HPS)

FPGA

Master Controller
System-on-Chip
(Intel/Altera)

LVDS

FPGA
(Intel)

REF_CLK

SPI

SLAVE DDS x
(AD9910)

SYN_IN

SYN_OUT

CH x

ADC
Multi-channel

DAC
Multi-channel

Serial_Data_In

SERDES
Rx_Core

8b/10b
Decoder

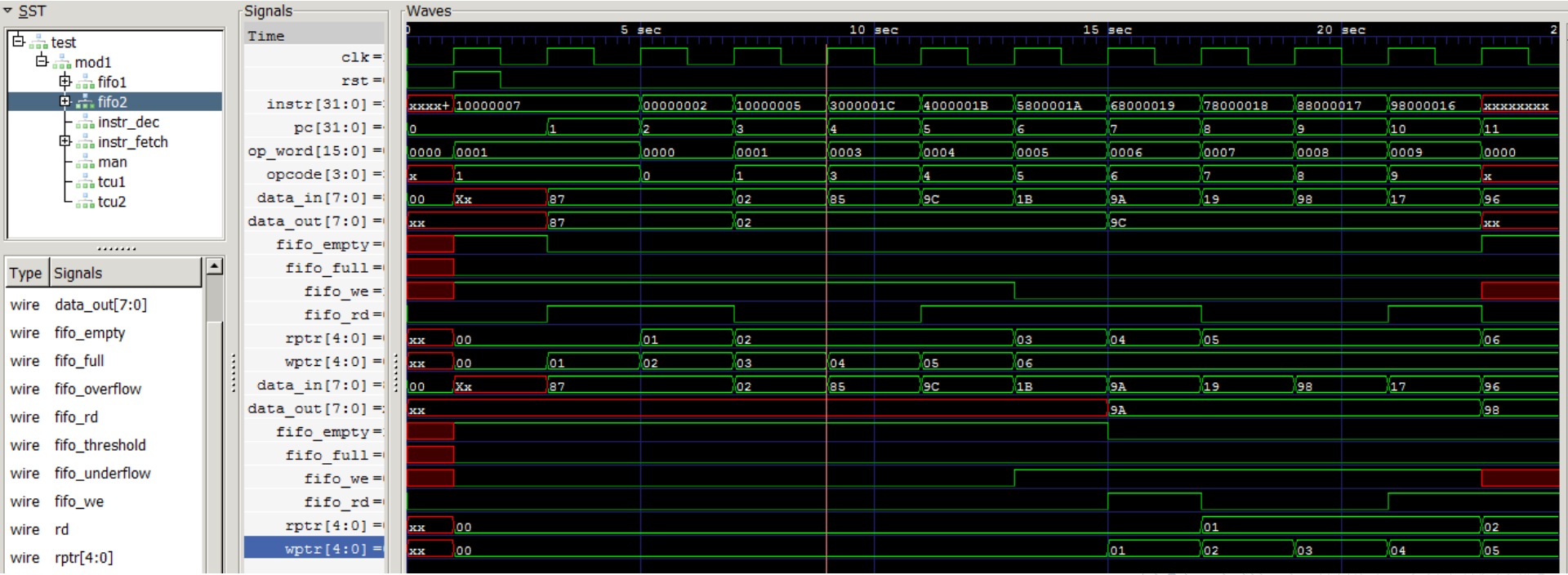
Instruction
Buffer

Timing Precise
Execution Unit

SPI Master

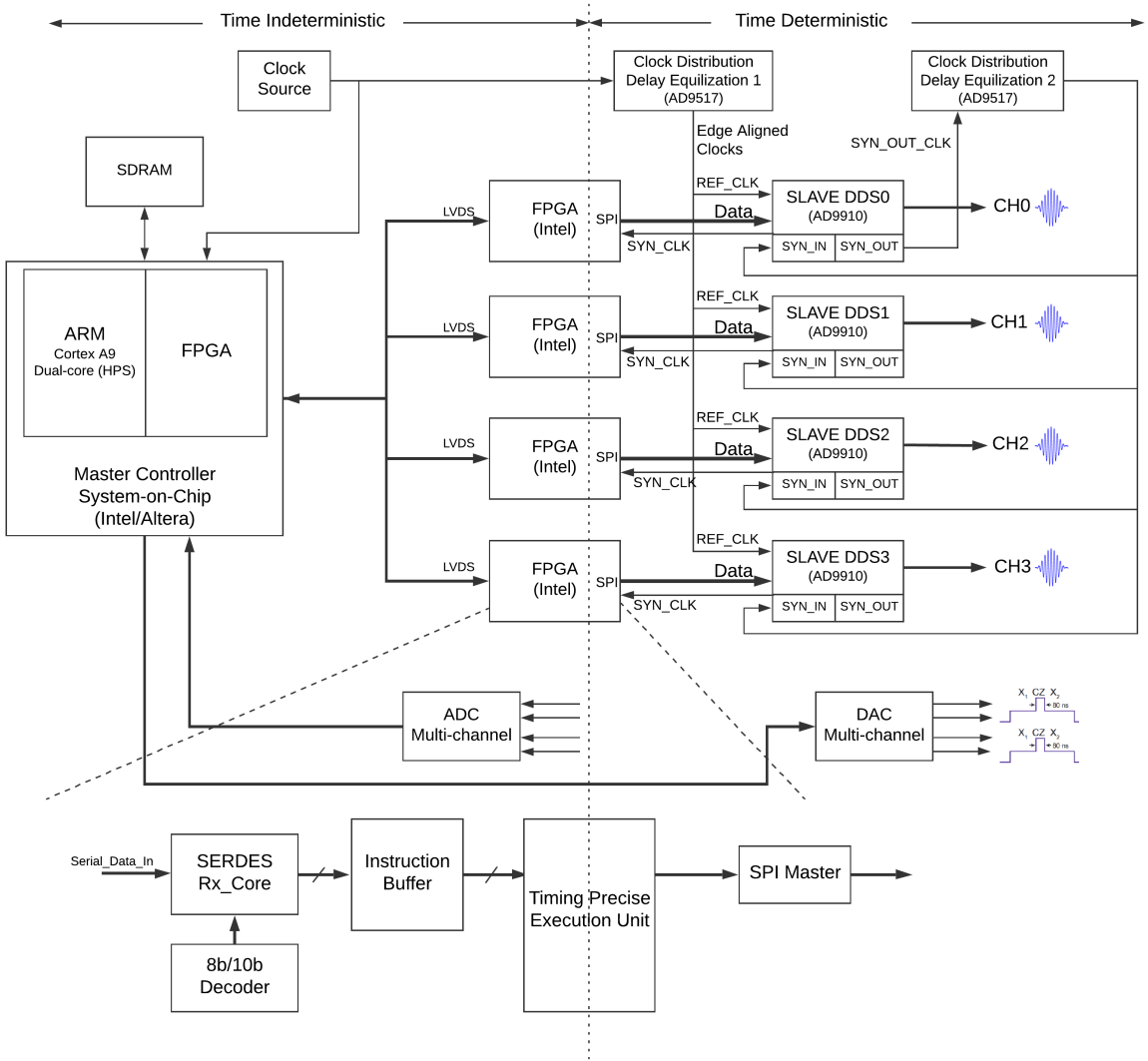
TU Delft

Timing Precise Execution Unit (using Synchronous FIFO, for simulation)



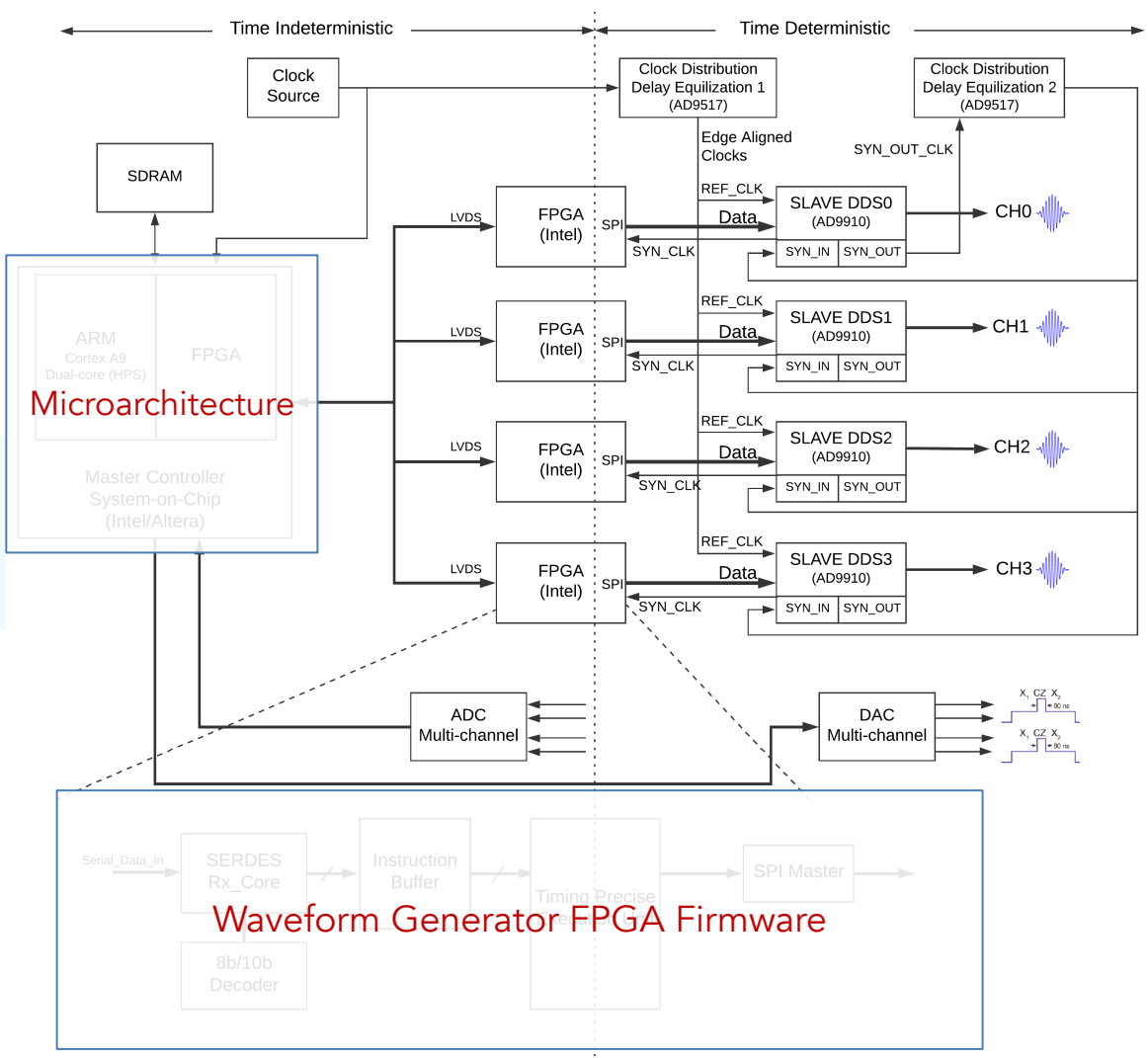
Timing Precise Execution Unit can be made Real-time, by implementing read and write statements to issue codewords/communicate payload data. Hence, no Async. FIFO Required.

A complete, distributed & synchronized microarchitecture for Waveform Generation

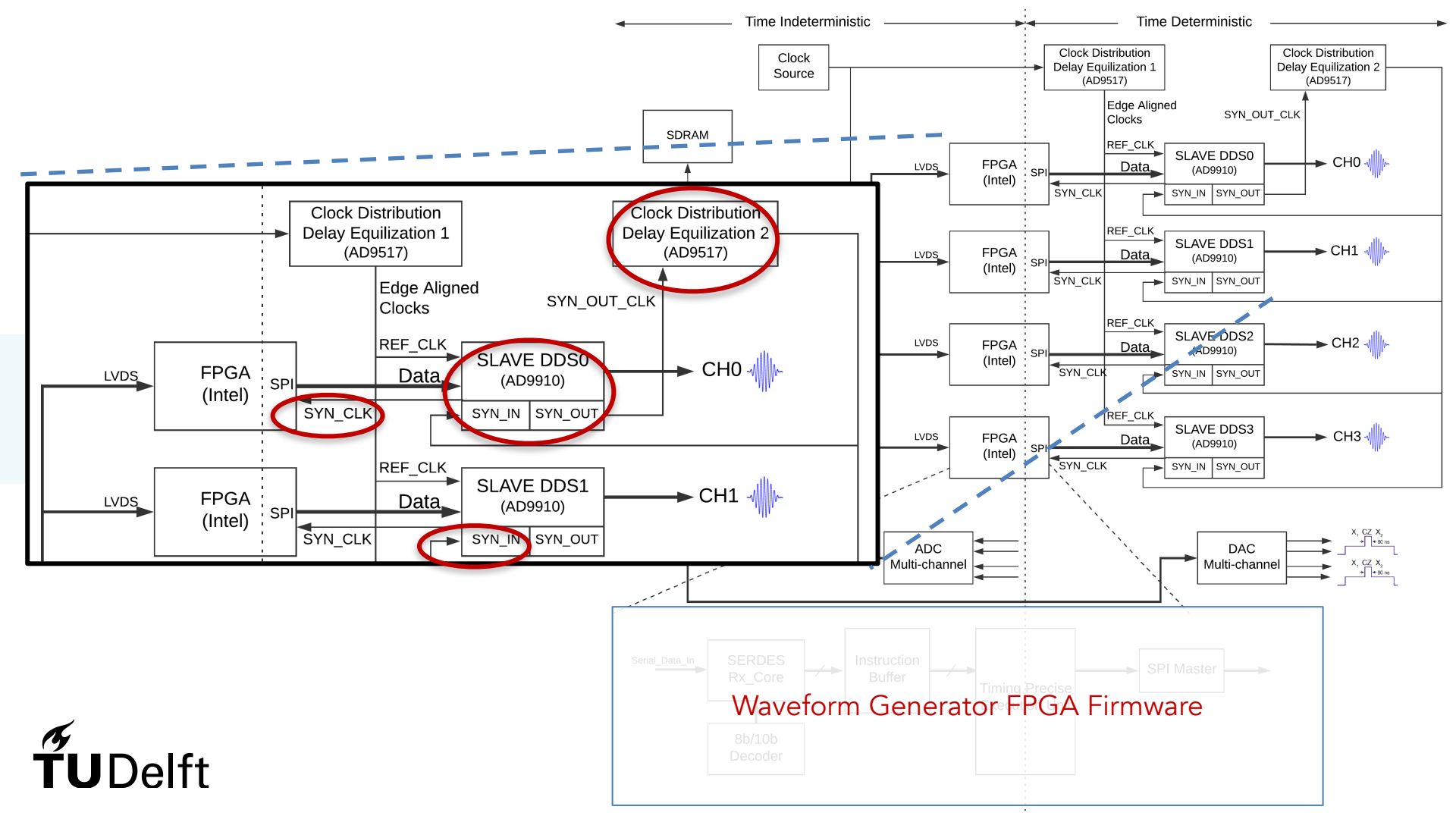


TU Delft

A complete, distributed & synchronized microarchitecture for Waveform Generation

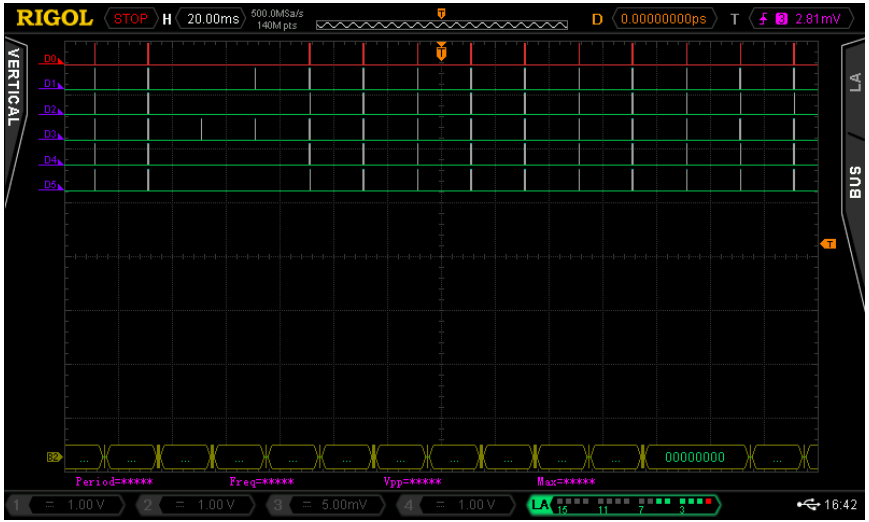


TU Delft

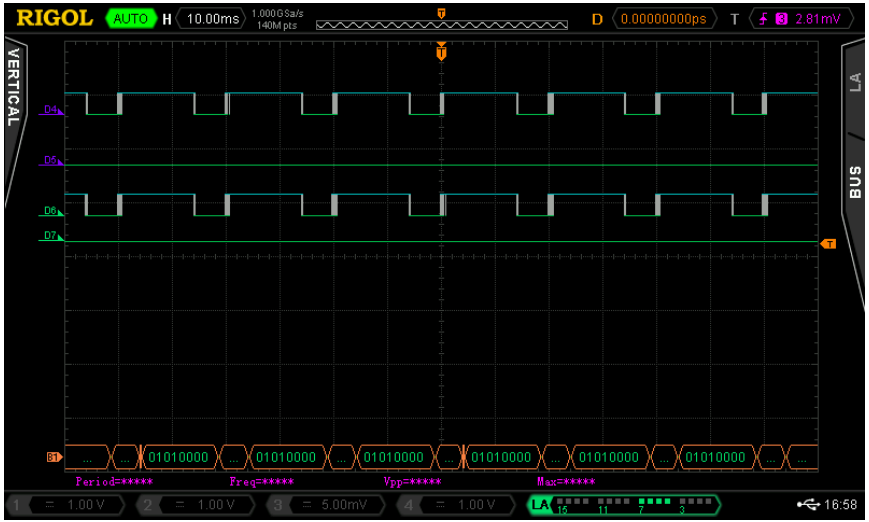

Time Indeterministic
Time Deterministic
Clock Source
Clock Distribution Delay Equilization 1 (AD9517)
Clock Distribution Delay Equilization 2 (AD9517)
SDRAM
Edge Aligned Clocks
SYN_OUT_CLK
REF_CLK
Data
SYN_CLK
SYN_IN
SYN_OUT
LVDS
SPI
FPGA (Intel)
SLAVE DDS0 (AD9910)
SLAVE DDS1 (AD9910)
SLAVE DDS2 (AD9910)
SLAVE DDS3 (AD9910)
CH0
CH1
CH2
CH3
ADC Multi-channel
DAC Multi-channel
Serial_Data_In
SERDES Rx_Core
Instruction Buffer
Timing Precise
SPI Master
8b/10b Decoder
Waveform Generator FPGA Firmware

TU Delft

Results from Hardware Testing



6 bit data_out 3.3V LVTTL single-ended signal from Master-Controller

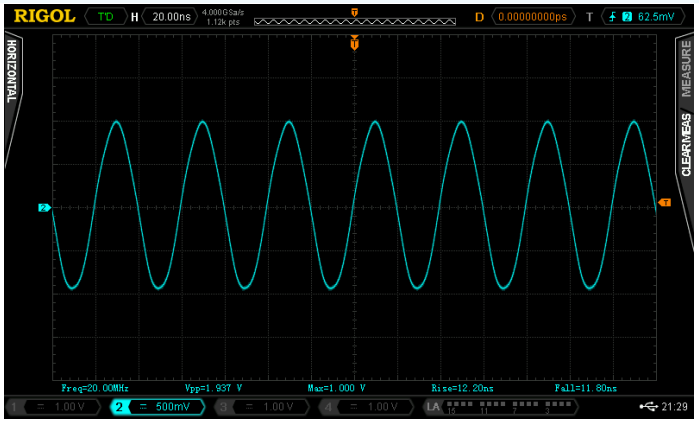


Parallel LVDS Clock and Data_Out Signals

Error Sources: Timing Mismatch in RX Clock and TX Clock; Global Clock Distribution.

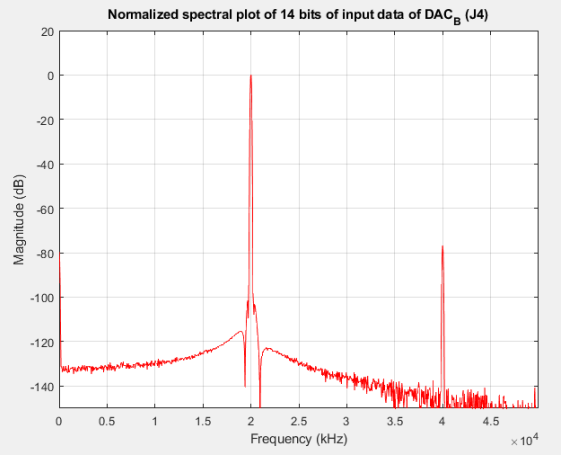
Mitigation: 2FF Synchronizer.

Output Waveforms from DAC-based AWG

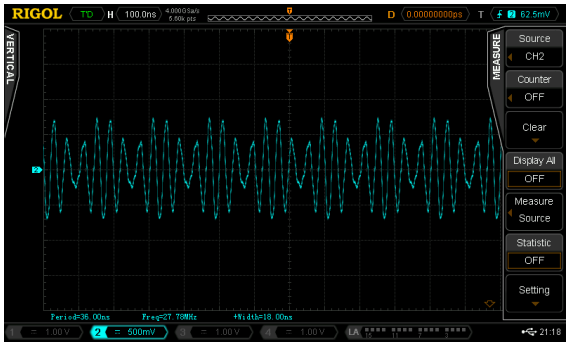


← 20MHz Sine Wave

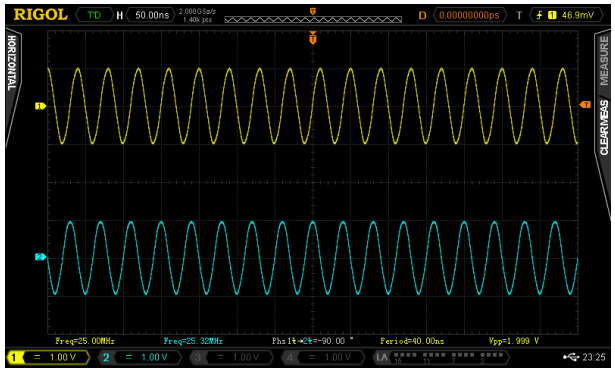
FFT Spectral Plot - 20MHz Sine Wave →



$$F_{NCO} = \left(\frac{F_{OSC}}{Accumulator}\right) (IncrementValue)$$



Arb. Waveform Generation using DAC with NCO on FPGA



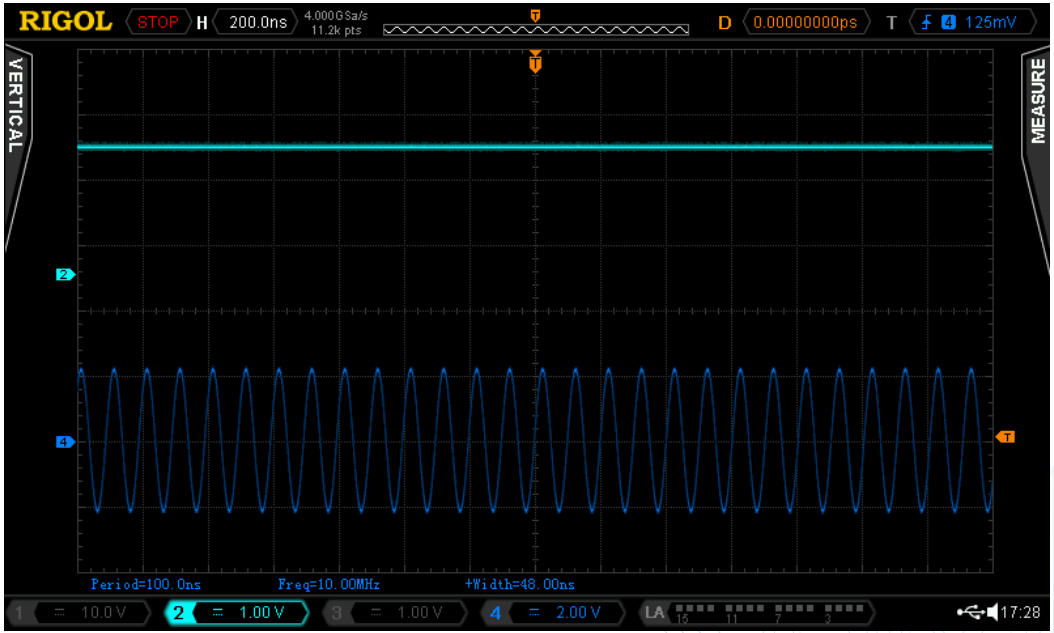
IQ Waveform Generation using DAC with NCO on

Resource Utilization	
Logic utilization (in ALMs)	628/41,910 (1 %)
Total block memory bits	163,328/5,662,720 (3 %)
Total RAM Blocks	18/553 (3 %)
Total PLLs	1/15 (7 %)

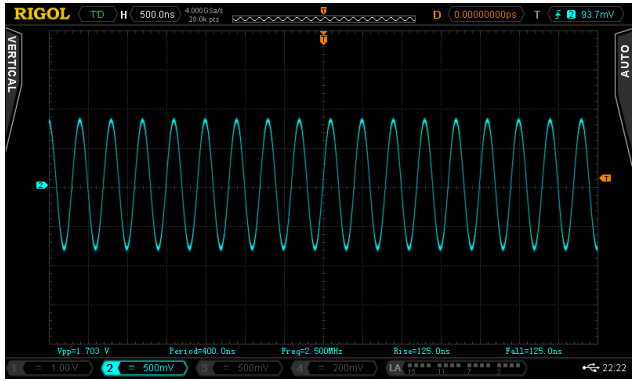
Output Waveforms from DAC-based AWG

Stored Waveform AWG Implementation on CBoxV3 Hardware

DC and Sine Wave (10MHz)



Output Waveforms from DDS-based AWG



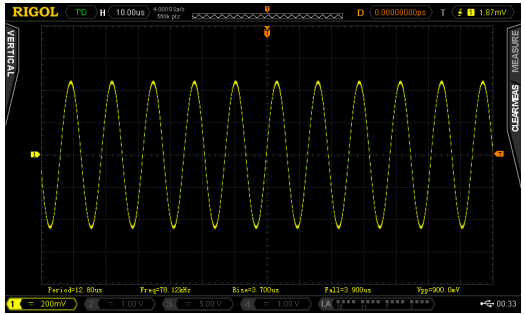
2.5MHz Sine Wave in Single-Tone Mode

Profile 0

Output Freq:	2.500000002	MHz
Phase Offset:	1 / 0.000	Deg
Amplitude SF:	1 / 0.00000	

Profile 1

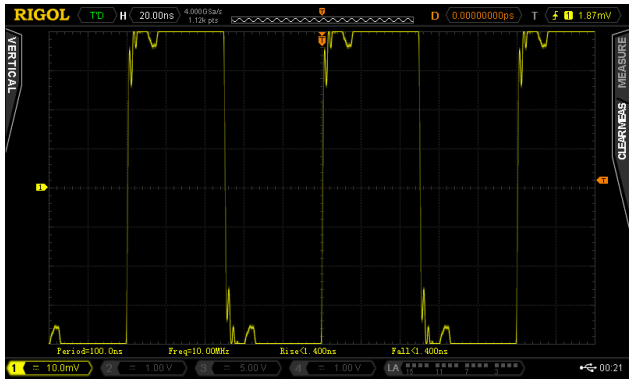
Output Freq:	30.000000005	MHz
Phase Offset:	1 / 0.0	Deg
Amplitude SF:	1 / 0.00000	



80KHz Sine Wave in RAM Mode

RAM Mode Amplitude values, sine wave ->

1	1187
2	2348
3	3459
4	4495
5	5433
6	6254
7	6939
8	7475
9	7848
10	8052
11	8081
12	7935
13	7618
14	7135
15	6498
16	5721



10MHz Square Wave in RAM Mode

RAM Segment 0

Beginning Address:	0
Final Address:	128
Address Step Rate:	3706.8800 µs
Mode Control:	Continuous Re-circulate
	☐ Zero Crossing ☐ No Dwell

Load RAM

RAM Segment 1

Beginning Address:	128
Final Address:	256
Address Step Rate:	0.0000 µs
Mode Control:	Continuous Re-circulate
	☐ Zero Crossing ☐ No Dwell

Conclusion

CC-Spin Quantum Microarchitecture			
Requirements Analysis (20%)	QISA Design (10%)	Hardware Configuration (40%)	Implementation & Testing(30%)

- A distributed, modular quantum micro-architecture for quantum control is developed – helps in management, analysis and scaling.
- We developed two versions of QISA for adding payload data (frequency, phase and amplitude) alongside eQASM.
- Real-time parameterized waveform 'synthesis' using DDS-DAC/NCO-based AWG.
- CTPG waveform generation using stored-waveform DAC-based AWG/RAM mode DDS-AWG.
- Replacing commercial AWGs with a more scalable alternatives – DDS, parameterized AWGs.
- High Speed Communication with modular AWG/DDS units via Mezzanine for clock synchronization and data communication.

TU Delft

Ready yet? Not Quite.

Future Work

(approx. timeline: 12-14 months)

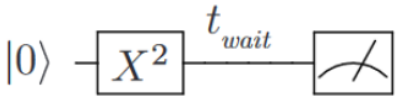
Priority

- Time-Stamp Generator for specifying duration and generate trigger.
- Multiple-Slave Hardware Synchronization with LVDS (via Mezzanine)/MGT (for performance).
- Implement the Measurement Unit.
- Implement Q Control Store/Microcode Unit and add Payload definitions.
- Verify Real-time Delays e.g. trigger to waveform generation delay.
- Overall Verification, Testing and Debugging. **<--- Ready for running simple SIMQ experiments.**
- SPI Master Controller for AD9910 DDS-DAC.
- Implement Version 1 or 2 of QISA for parallelized operation of >8 qubits.
- Benchmarking

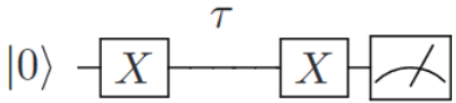
TU Delft

Now done? Almost.

Quantum Benchmark? What experiments can be run?

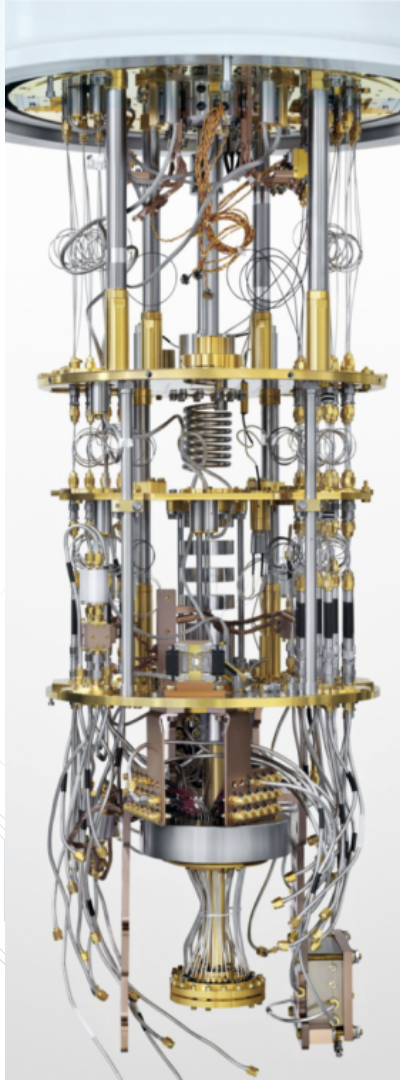


Measure Spin relaxation time is measured by preparing the qubit to spin-up and varying the wait time before readout.



Ramsey Experiment: Measure dephasing time by applying Ramsey pulse sequence and vary the free evolution time, τ.

Randomized Benchmarking: Measure fidelity of single qubit operations. Sequences of 2 to 2000 single qubit gates, all solving to identity.



TU Delft

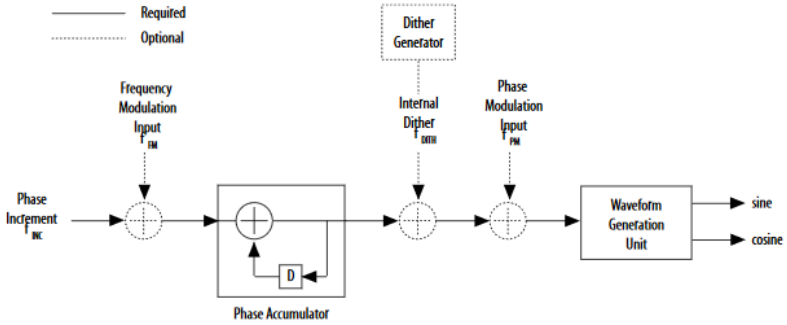
fin.

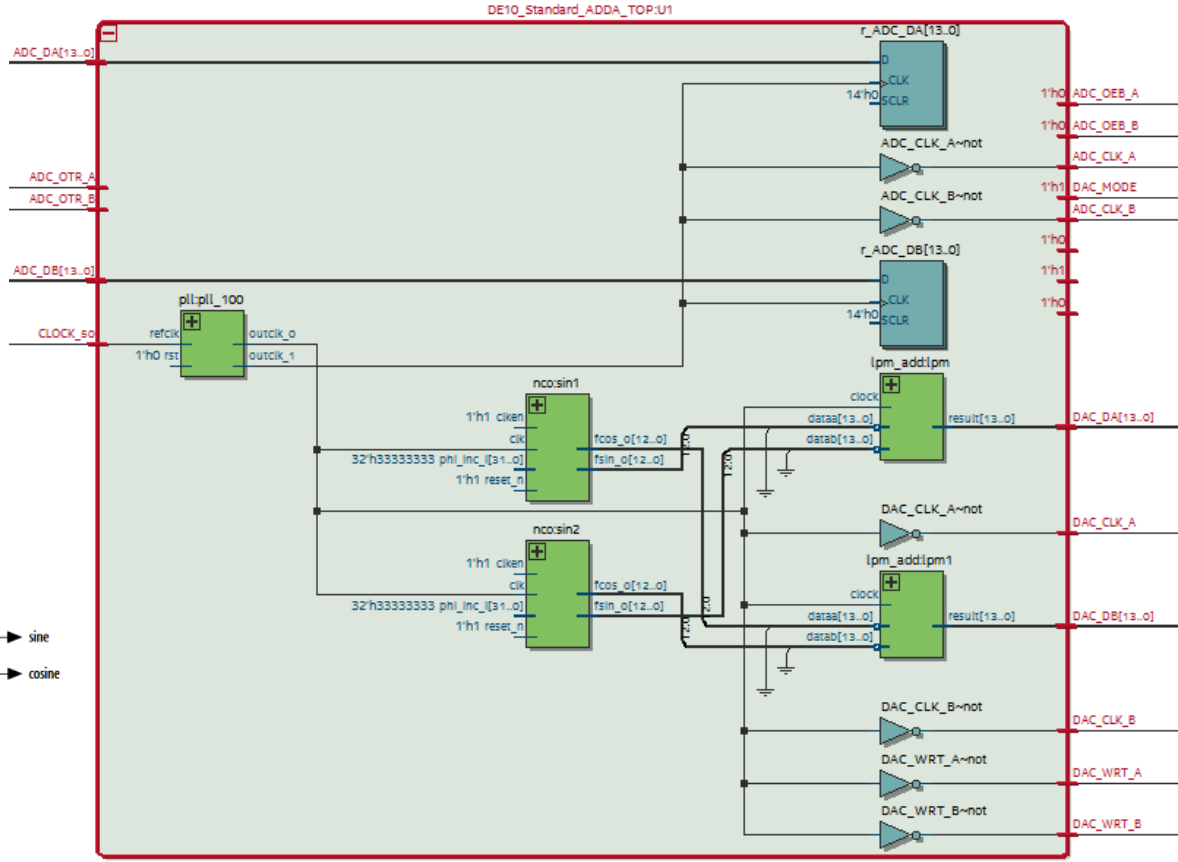
"Nature isn't classical, dammit; and if you want to make a simulation of nature, you'd better make it quantum mechanical, and by golly, it's a wonderful problem, because it doesn't look so easy."

Richard P. Feynman

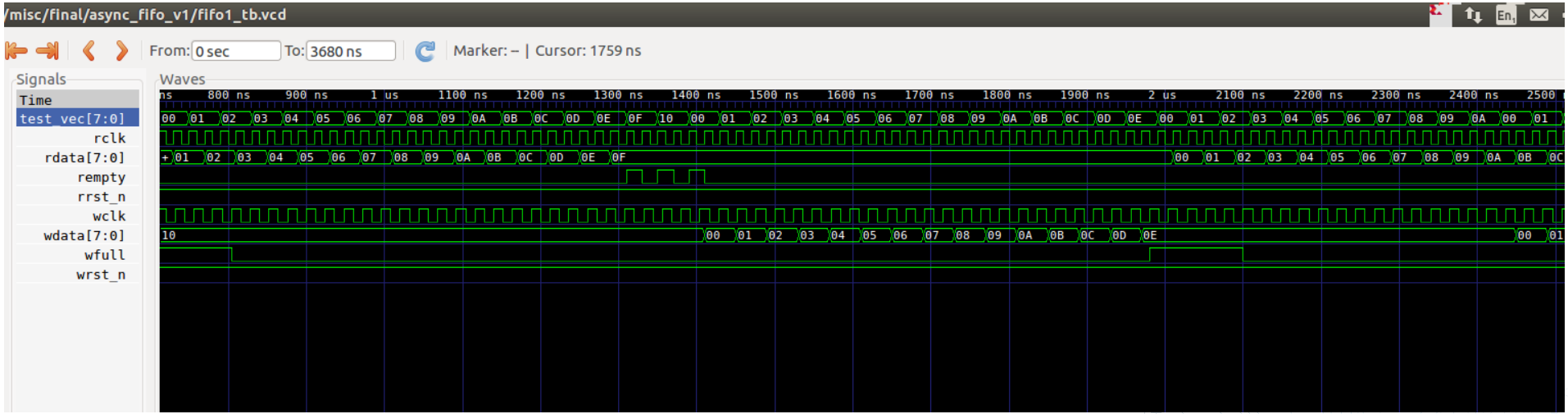
Extra Slides.

NCO-based Waveform Generation using AWG





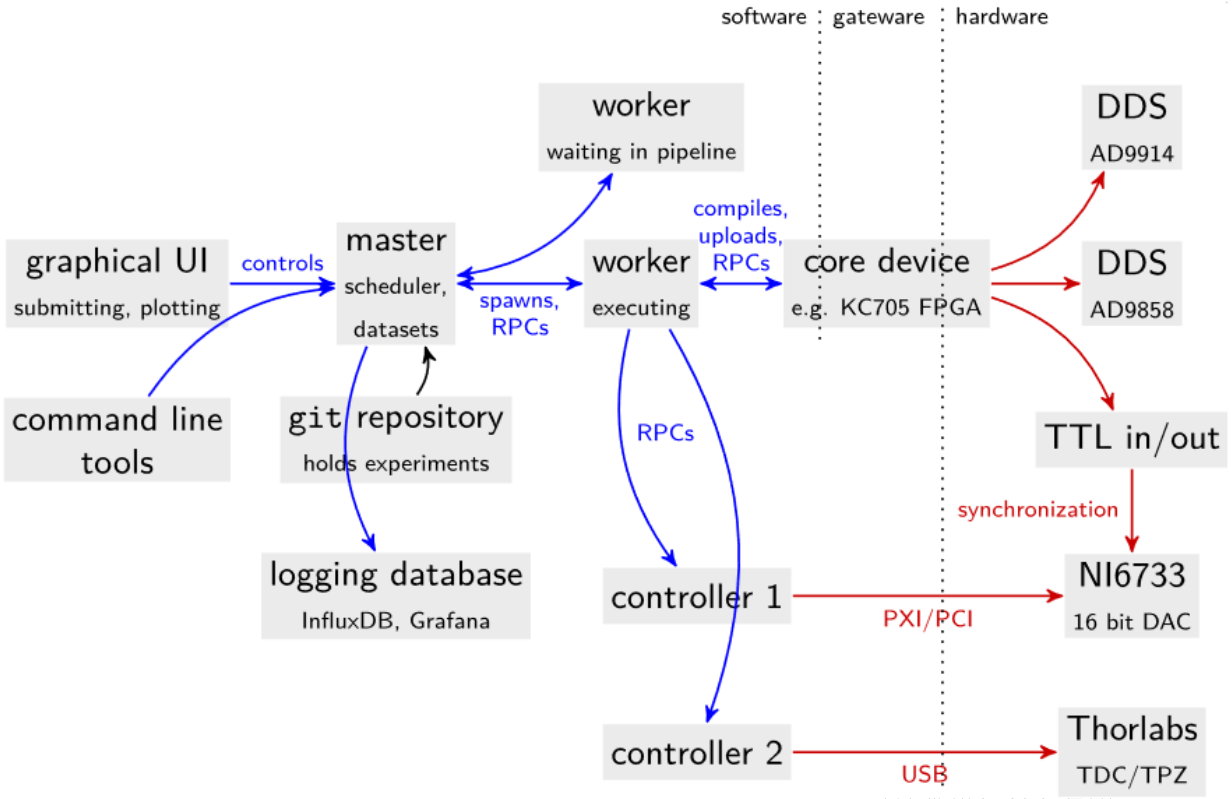
Asynchronous FIFO



State-of-the-Art Systems

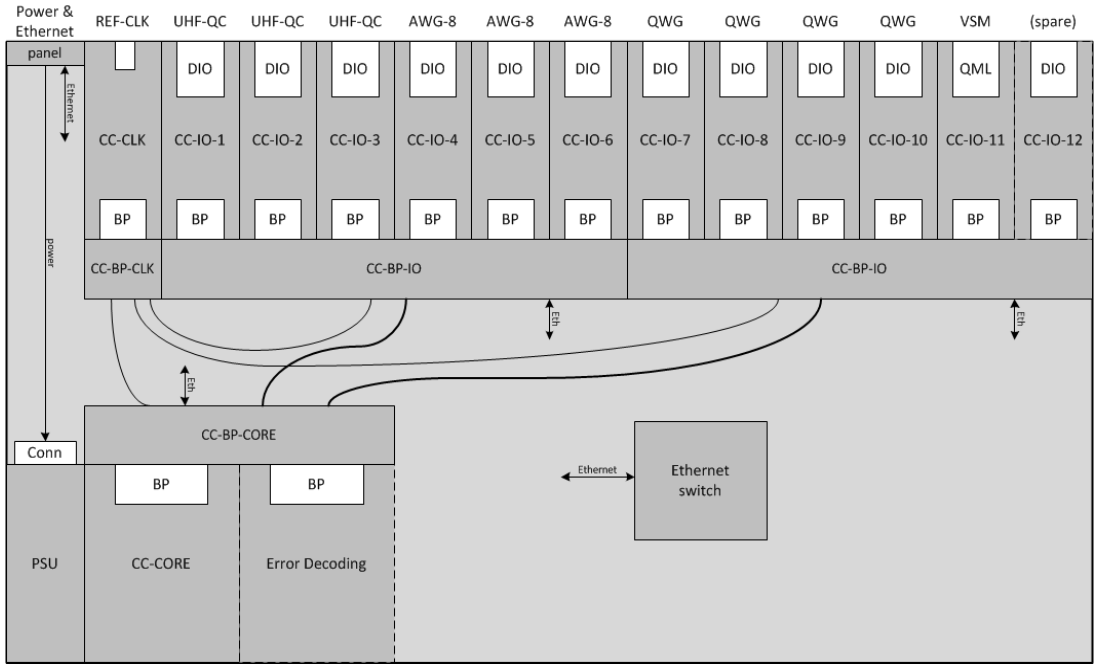
Sinara Ecosystem

(for quantum optics and ion-traps)



State-of-the-Art Systems

QuTech Control Architecture



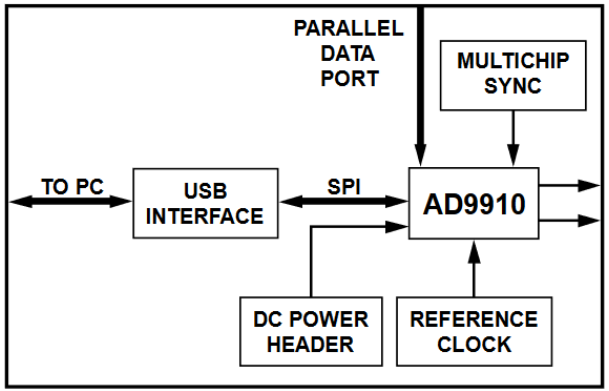
One **CC-IO module** per connected instrument.

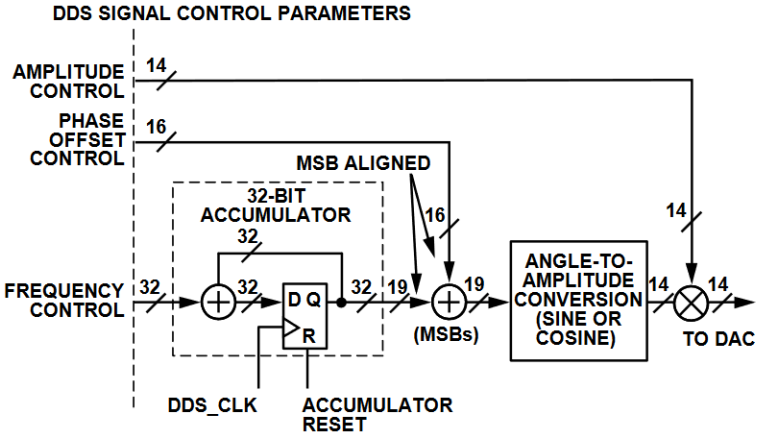
Star network @ 2.4Gb/s (each direction) between CC-IO's for low-latency communication between modules.

One **CC-CORE**:
Hub for low-latency communication between CC-IO's.

Main TCP/IP entry point to the external world.

AD9910 DDS-DAC





Timing Control Unit