Computational and Communication Architectures for Modular Multilevel Converter Construction

A thesis submitted to The University of Manchester for the degree of

Doctor of Philosophy

in the Faculty of Science and Engineering

2022

Jack Andrews

Department of Electrical and Electronic Engineering

Blank Page

Table of Contents

Table of Contents **3**

List of Figures **8**

List of Tables **12**

List of Source Code Listings **13**

Nomenclature **14**

Abstract **21**

Declaration **22**

Copyright Statement **22**

Acknowledgements **23**

1 Introduction **24**

1.1 Background 24

1.1.1 Offshore Wind Farm Grid Connections 24

1.1.2 Renewable Electricity Generation Integration Challenges 26

1.1.3 Voltage Source Converter HVDC 27

1.2 Aims and Objectives 29

1.3 Main Contributions 29

1.4 Thesis Structure 31

2 MMC Structure and Control **34**

2.1 Circuit Structure 34

2.1.1 Selection of Nominal Submodule Capacitor Voltage 37

2.2 Control Structure 39

2.2.1 dq Current Control 40

2.2.2 High-level (PQ) Control 43

2.2.3 Low-level Control 43

2.3 Summary 47

3 Capacitor Balancing Control **48**

3.1 Terminology 48

3.1.1 Modulation Algorithm 49

3.1.2 Capacitor Balancing Control Method 49

3.1.3 Sorting Algorithm 50

3.2 Review of Capacitor Balancing Control Methods 50

3.2.1 Sorting Algorithm Requirement 50

3.2.2 Sampling Period 54

3.3 Summary 57

4 Sorting Algorithms 58

4.1 Comparison Metrics 58

4.1.1 Bachmann-Landau (Big-*O*) Notation 59

4.1.2 Time Complexity 60

4.1.3 Space Complexity 61

4.1.4 Sequential vs. Parallel Sorting Algorithms 62

4.2 Sorting Networks 63

4.3 Summary 64

5 Simulation Model Overview 65

5.1 Circuit Structure 65

5.2 Control Structure 67

5.3 Capacitor Balancing Control Custom Component 69

5.3.1 Capacitor Balancing Control Method Implementation 71

5.4 Summary 74

6 Sorting Algorithms for Capacitor Balancing Control 75

6.1 Motivations 75

6.1.1 Sorting Algorithm Execution Time 76

6.1.2 Sorting Algorithm Implementation Resource Usage 76

6.2 Sorting Algorithm Implementation 77

6.2.1 Control Hardware Targets 77

6.2.2 Programming Methods 78

6.2.3 Sorting Algorithm Implementation Summary 80

6.3 Sorting Algorithm Execution Time Range 83

6.3.1 Real-Time CPU Target 85

6.3.2 FPGA Target 86

6.4 Comparison of Balancing Method Execution Time Window and Sorting Algorithm Execution Time 87

6.4.1 Terminology 88

6.4.2 Selection of Maximum Allowable Sample Delay 90

6.4.3 Execution Time Results 91

6.5 Sorting Algorithm FPGA Logic Resource Usage 102

6.5.1 FPGA Synthesis Process 102

6.5.2 Logic Resource Usage Results 106

6.6 Summary 112

7 Converter Hardware Prototype 114

7.1 Design Objectives 114

7.2 Hardware Overview 115

7.2.1 Submodule Overview 117

7.2.2 Hardware Modifications 120

7.3 Control Architecture Overview 121

7.3.1 Human-Machine Interface (HMI) 124

7.3.2 Local Area Network (LAN) 124

7.3.3 Converter Control Unit (CCU) 124

7.3.4 Data Acquisition (DAQ) 125

7.3.5 Phase Control Unit (PCU) 125

7.3.6 Arm Control Unit (ACU) 125

7.3.7 Local Controller (LC) 126

7.4 Summary 127

8 Control System Software Development 128
8.1 Overview of the LabVIEW Programming Language 128
8.2 Review of Existing CHP Control Software 130
8.3 Design Objectives 131
8.4 PXIe System Internal Architecture 131
8.4.1 CCU Host Communication Buses 132
8.4.2 Clock Sources 133
8.4.3 Other Connections 134
8.5 Controller Sample Clock Generation 135
8.5.1 PXI_Clk10 Clock Divider Block Operation 137
8.5.2 Event-Driven Mode Value Change Event Sources 140
8.6 Hardware Delay Measurement 143
8.7 Data Acquisition and Logging 144
8.7.1 ACU Capacitor Voltage Data Transfer to CCU 145
8.7.2 Data Transfer to HMI PC 145
8.7.3 Data Plotting and Logging on HMI PC 145
8.7.4 Measurement Timestamping 146
8.8 Code Optimisation 147
8.9 Summary 148
9 Conclusion and Future Work 150
9.1 Conclusion 150
9.2 Future Work 153
9.2.1 Capacitor Balancing Control Loop 153
9.2.2 Converter Hardware Prototype 155
References 157
Appendix A - Pseudocode for Sorting Algorithms 166
A-1 Bubble Sort 166
A-2 Insertion Sort 167

A-3 Merge Sort 168

A-4 Quick Sort..170

Appendix B - Source Code for Custom PSCAD/EMTDC Component.......................172

B-1 Common Source Code...173

B-2 Periodic CBC Method Source Code..176

B-3 Average Tolerance Band CBC Method Source Code 176

B-4 Cell Tolerance Band CBC Method Source Code 176

B-5 Bubble Sort Subroutine Source Code..177

Appendix C - Selection of Maximum Allowable Control Cycle Delay Threshold 179

Appendix D - Supplementary Plots for Chapter 6...188

D-1 Scenario 1: Steady State Power Import...189

D-2 Scenario 2: Power Flow Reversal 195

D-3 Scenario 3: AC 3 Phase-ground Fault 201

Appendix E - CHP Sensing and Measurement Circuit Modifications........................207

List of Figures

Figure 1.1: Subsea HVAC vs. HVDC transmission scheme break-even distance 25
Figure 2.1: Circuit diagram of (a) modular multilevel converter, and (b) a half-bridge submodule 35
Figure 2.2: Modular multilevel converter equivalent circuit for a single phase 36
Figure 2.3: MMC cascaded control system overview 39
Figure 2.4: MMC connection to AC system, one phase shown 40
Figure 2.5: State feedback system block diagrams for MMC dq current control loop 42
Figure 2.6: MMC low-level control structure 43
Figure 2.7: Equivalent circuit for a single phase of an MMC 44
Figure 3.1: Block diagram showing relationship between modulation algorithm, CBC method, and sorting algorithm 49
Figure 3.2: CBC method taxonomy based upon sorting algorithm requirements 51
Figure 3.3: CBC method taxonomy based upon CBC loop sampling period properties 55
Figure 4.1: Common growth functions for sorting algorithm execution time and resource usage 59
Figure 4.2: Compare-swap operator 62
Figure 4.3: Categorisation of sorting algorithms and processing hardware on sequential vs. parallel sliding scale 63
Figure 5.1: MMC model circuit diagram 66
Figure 5.2: Block diagram showing MMC model control structure 67
Figure 5.3: PSCAD/EMTDC custom capacitor balancing control component mask 69
Figure 5.4: Flowchart showing periodic capacitor balancing control method sequence of operations 72
Figure 5.5: Flowchart showing trigger decision tree for average tolerance band CBC method 73
Figure 5.6: Flowchart showing trigger decision tree for cell tolerance band CBC method 74
Figure 6.1: Execution time range as a function of number of inputs vs. sorting algorithm 84
Figure 6.2: Timing diagram showing the relationship between $T_{s-control}$, T_{s-CBC}, t_{sort}, $M_{control}$, M_{CBC}, against sampling instant number, k 88
Figure 6.3: Timing diagram showing the relationship between $M_{control}[k]$, $M_{control-thresh}$, and total number of CBC loop sampling instants 91
Figure 6.4: Converter-level response for scenario 1 93

Figure 6.5: Percentage of CBC loop cycles where $M_{control-thresh}$ = 8 was exceeded for scenario 1 94
Figure 6.6: Converter-level response for scenario 2 95
Figure 6.7: Percentage of CBC loop cycles where $M_{control-thresh}$ = 8 was exceeded for scenario 2 96
Figure 6.8: Phase A, upper arm current for scenario 2 96
Figure 6.9: Converter-level response for scenario 3 98
Figure 6.10: Percentage of CBC loop cycles where $M_{control-thresh}$ = 8 was exceeded for scenario 3 99
Figure 6.11: Phase A, upper arm current for scenario 3 100
Figure 6.12: Phase A, upper arm SM insertion index for scenario 3 100
Figure 6.13: Synthesis process for sorting algorithm logic resource usage measurements 103
Figure 6.14: Plot of total number of slices used vs. number of inputs vs. programming method 107
Figure 6.15: Plot of total number of slices used vs. number of inputs vs. sorting algorithm 110
Figure 7.1: Annotated CHP system hardware layout, front (left) and rear (right) 116
Figure 7.2: CHP submodule simplified schematic 118
Figure 7.3: Four SM valve ‘stack’ and local controller (left) and individual SM (right) 120
Figure 7.4: CHP distributed control architecture overview 123
Figure 7.5: NI PXIe system with CCU, DAQ, ACU, and PCU cards (left to right) 126
Figure 7.6: Terasic DE-10 Nano Local Controller (left), with breakout board installed (right) 126
Figure 8.1: CHP PXIe system internal architecture (existing control software) 132
Figure 8.2: CHP PXIe system internal architecture (new control software) 135
Figure 8.3: Controller sample clock generator functional overview 136
Figure 8.4: *PXI_Clk10* clock divider block architecture 137
Figure 8.5: Timing diagram of global PXIe system synchronisation signals and PCU/ACU FPGA internal signals 137
Figure 8.6: Timing diagram showing operation of the clock divider block preload input terminal 139
Figure 8.7: External reference voltage PLL sample clock sources 140
Figure 8.8: Circulating current suppression control loop sample clock sources 141

Figure 8.9: Output voltage control loop sample clock sources 141
Figure 8.10: Arm balancing and NLC sample clock sources 142
Figure 8.11: Capacitor balancing control loop sample clock sources 142
Figure 8.12: Firing signal generation loop sample clock sources 143
Figure 8.13: CHP control loop hardware delay measurement set up 144
Figure 8.14: NLC loop timing data acquisition from logic analyser 144
Figure 8.15: Screen capture of HMI PC user interface showing CHP SM capacitor voltage plots using data transferred from ACU FPGA 146
Figure C-1: PSCAD/EMTDC MMC model e^{-sT} delay block location 183
Figure C-2: Converter level response, steady state operation at P^* = −1200 MW, Q^* = −400 MVAr (import, inverter mode) 184
Figure C-3: Maximum instantaneous capacitor voltage deviation from $V_{cap-nom}$ as a percentage of $V_{cap-nom}$ at different delay lengths 185
Figure C-4: Zoomed-in plot of maximum instantaneous deviation from $V_{cap-nom}$ as a percentage of $V_{cap-nom}$ at different delay lengths (top), and phase A upper arm current (bottom, left-hand y-axis) and phase A upper arm SM insertion index (bottom, right-hand y-axis) 186
Figure D-5: Upper arm electrical quantities, scenario 1, periodic CBC 189
Figure D-6: Sorting algorithm missed control cycle histograms, scenario 1, periodic CBC 190
Figure D-7: Upper arm electrical quantities, scenario 1, average tolerance band CBC 191
Figure D-8: Sorting algorithm missed control cycle histograms, scenario 1, average tolerance band CBC 192
Figure D-9: Upper arm electrical quantities, scenario 1, cell tolerance band CBC 193
Figure D-10: Sorting algorithm missed control cycle histograms, scenario 1, cell tolerance band CBC 194
Figure D-11: Upper arm electrical quantities, scenario 2, periodic CBC 195
Figure D-12: Sorting algorithm missed control cycle histograms, scenario 2, periodic CBC 196
Figure D-13: Upper arm electrical quantities, scenario 2, average tolerance band CBC .. 197
Figure D-14: Sorting algorithm missed control cycle histograms, scenario 2, average tolerance band CBC 198
Figure D-15: Upper arm electrical quantities, scenario 2, cell tolerance band CBC 199

Figure D-16: Sorting algorithm missed control cycle histograms, scenario 2, cell tolerance band CBC 200
Figure D-17: Upper arm electrical quantities, scenario 3, periodic CBC 201
Figure D-18: Sorting algorithm missed control cycle histograms, scenario 3, periodic CBC. 202
Figure D-19: Upper arm electrical quantities, scenario 3, average tolerance band CBC ..203
Figure D-20: Sorting algorithm missed control cycle histograms, scenario 3, average tolerance band CBC 204
Figure D-21: Upper arm electrical quantities, scenario 3, cell tolerance band CBC 205
Figure D-22: Sorting algorithm missed control cycle histograms, scenario 3, cell tolerance band CBC 206

List of Tables

Table 2.1: Half-bridge submodule switch states 36
Table 4.1: Best- and worst-case time complexities for several common sorting algorithms for sorting-based CBC methods 60
Table 4.2: Worst-case space complexities for several common sorting algorithms for sorting-based CBC methods 61
Table 5.1: MMC model parameters 66
Table 6.1: Summary of key specifications for control hardware targets 77
Table 6.2: Comparison of sorting algorithm programming methods 80
Table 6.3: Matrix of sorting algorithms, control hardware targets and programming languages tested 82
Table 6.4: List of sorting algorithms and programming method combinations tested 91
Table 6.5: Total number of CBC loop sampling instances per CBC method, scenario 1 ... 94
Table 6.6: Total number of CBC loop sampling instances per CBC method, scenario 2 ... 97
Table 6.7: Total number of CBC loop sampling instances per CBC method, scenario 3 . 101
Table 7.1: CHP nominal electrical ratings (present configuration) 115
Table 7.2: List of CHP auxiliary hardware and functions 117
Table 7.3: CHP submodule electrical ratings 118
Table 7.4: CHP submodule component key specifications 119
Table 8.1: Summary of CHP control hardware targets and LabVIEW variants 129
Table 8.2: CHP LabVIEW FPGA control software FPGA slice resource usage summary 147

List of Source Code Listings

Listing A-1: Bubble Sort algorithm pseudocode 166
Listing A-2: Insertion Sort algorithm pseudocode..167
Listing A-3: Merge Sort algorithm pseudocode..169
Listing A-4: Quick Sort algorithm pseudocode 171
Listing B-1: Custom component common FORTRAN source code, part 1 174
Listing B-2: Custom component common FORTRAN source code, part 2 175
Listing B-3: Periodic CBC method FORTRAN source code 176
Listing B-4: Average Tolerance Band CBC method FORTRAN source code..................176
Listing B-5: Cell Tolerance Band CBC method FORTRAN source code 176
Listing B-6: Bubble Sort algorithm FORTRAN source code, ascending sort direction....177
Listing B-7: Bubble Sort algorithm FORTRAN source code, descending sort direction..178

Nomenclature

List of Acronyms

Acronym	**Definition**
AC	Alternating Current
ACU	Arm Control Unit
ADC	Analogue-to-Digital Converter
ATB	Average Tolerance Band (capacitor balancing control method)
AVC	Arm Voltage Control
CAGR	Compound Annual Growth Rate
CBC	Capacitor Balancing Control
CCSC	Circulating Current Suppression Control
CCU	Converter Control Unit
CHP	Converter Hardware Prototype
CPU	Central Processing Unit
CSC	Current Source Converter
CTB	Cell Tolerance Band (capacitor balancing control method)
CTBoptimised	Cell Tolerance Band Optimised (capacitor balancing control method)
CTBsequence	Cell Tolerance Band with Sequence Reversing (capacitor balancing control method)
DAQ	Data Acquisition
DC	Direct Current
DMA	Direct Memory Access
DSP	Digital Signal Processor
EMT	Electromagnetic Transient
ESO	Electricity System Operator
FB-SM	Full-Bridge Submodule

Acronym	**Definition**
FF	Flip-Flop
FIFO	First-In First-Out
FOBB	Fibre Optic Breakout Board
FPGA	Field Programmable Gate Array
FSM	Finite-State Machine
GB/s	Gigabytes per second
GUI	Graphical User Interface
HB-SM	Half-Bridge Submodule
HDL	Hardware Description Language
HMI	Human-Machine Interface
HVAC	High Voltage Alternating Current
HVDC	High Voltage Direct Current
IGBT	Insulated Gate Bipolar Transistor
IO	Input-Output
KVL	Kirchoff’s Voltage Law
LAN	Local Area Network
LC	Local Controller
LUT	Lookup Table
MB/s	Megabytes per second
Mbps	Megabit per second
MMC	Modular Multilevel Converter
MOSFET	Metal-Oxide-Semiconductor Field-Effect Transistor
MPI	Multi-Purpose Interconnector
MSa/s	Mega samples per second
MT-HVDC	Multi-Terminal High Voltage Direct Current

Acronym	**Definition**
NI	National Instruments
NLC	Nearest Level Control
OS	Operating System
OTNR	Offshore Transmission Network Review
P	Proportional (control)
PCB	Printed Circuit Board
PCC	Point of Common Coupling
PCI	Peripheral Component Interconnect
PCIe	PCI Express
PCU	Phase Control Unit
PD-PWM	Phase Disposition Pulse-Width Modulation
PI	Proportional Integral
PIB	Power Interface Board
PLL	Phase-Locked Loop
PR	Proportional Resonant
PSC-PWM	Phase-Shifted Carrier Pulse-Width Modulation
PV	Photo-Voltaic
PWM	Pulse-Width Modulation
PXI	PCI eXtensions for Instrumentation
PXIe	PCIe eXtensions for Instrumentation
RAM	Random-Access Memory
RSF	Reduced Switching Frequency
RT	Real-Time
RTL	Register Transfer Level
RTOS	Real-Time Operating System

Acronym	**Definition**
SCR	Short-Circuit Ratio
SCTL	Single-Cycle Timed Loop
SM	Submodule
SMID	Submodule Identifier
SoC	System-on-Chip
SPI	Serial Peripheral Interface
TSO	Transmission System Operator
UART	Universal Asynchronous Receiver-Transmitter
VCO	Voltage-Controlled Oscillator
VI	Virtual Instrument
VSC	Voltage Source Converter

List of Symbols

Symbol	**Definition**	**S.I. Units**
C_{SM}	Submodule capacitance	F
f_0	AC system fundamental frequency	Hz
f_{clock}	FPGA clock frequency	Hz
$f_{s-control}$	Control cycle sampling frequency $(1/T_{s-control})$	Hz
f_{s-CBC}	Capacitor balancing control method sampling frequency $(1/T_{s-CBC})$	Hz
f_{s-MOD}	Modulation algorithm sampling frequency $(1/T_{s-MOD})$	Hz
I_{dc}	DC bus current	A
$I_{ac(abc)}$	AC output current, per phase	A
$I_{u(abc)}$	Upper arm current, per phase	A
$I_{l(abc)}$	Lower arm current, per phase	A
$I_{circ(abc)}$	Circulating current, per phase	A
$I_{diff(abc)}$	Difference current	A
I_{DS}	MOSFET drain-source current	A
k	Sampling instant number	–
L_{arm}	Arm inductance	H
$m_{a-(abc)}$	Modulation index, per phase	–
$M_{control}$	Number of samples delay due to sorting algorithm, relative to control cycle sampling period	–
M_{CBC}	Number of samples delay due to sorting algorithm, relative to capacitor balancing control loop sampling period	–
N_{SM}	Number of submodules per arm	–
N_{level}	Number of output voltage levels	–
$N_{onu(abc)}$	Upper arm submodule insertion index, per phase	–
$N_{onl(abc)}$	Lower arm submodule insertion index, per phase	–

Symbol	**Definition**	**S.I. Units**
n	Number of array elements to sort	−
N_{slice}	Total number of FPGA slices used	−
P	Active power	W
p	Derivative operator (d/dt)	−
Q	Reactive power	VAr
R_{arm}	Arm resistance	Ω
R_{load}	Load resistance	Ω
$R_{DS(on)}$	MOSFET on resistance	Ω
R_{dis}	Discharge resistance	Ω
S	Apparent power	VA
S_T	Transformer apparent power	VA
T_{s-MOD}	Modulation algorithm sampling period ($1/f_{s-MOD}$)	s
T_{s-CBC}	Capacitor balancing control method sampling period ($1/f_{s-CBC}$)	s
$T_{s-control}$	Control cycle sampling period ($1/f_{s-control}$)	s
t_{sort}	Sorting algorithm execution time	s
V_{dc}	DC bus voltage (pole-pole)	V
$V_{ac(abc)}$	AC phase-neutral voltage, per-phase	V
$V_{s(abc)}$	AC system voltage at point of common coupling, per phase	V
$V_{c(abc)}$	Internal converter voltage, per phase	V
V_N	AC network voltage, per phase	V
$V_{diff(abc)}$	Difference voltage	V
$V_{ext(abc)}$	External voltage reference, per phase	V
V_{SM}	Submodule output voltage	V
$V_{u(abc)}$	Upper arm voltage, per phase	V

Symbol	**Definition**	**S.I. Units**
$V_{l(abc)}$	Lower arm voltage, per phase	V
$V_{capu(abc)}^{i}$	Upper arm capacitor voltage, ith capacitor, per phase	V
$V_{capl(abc)}^{i}$	Lower arm capacitor voltage, ith capacitor, per phase	V
$V_{cap-nom}$	Nominal submodule capacitor voltage	V
$V_{capu,l}^{\Sigma}(t)$	Instantaneous sum capacitor voltage	V
$\overline{V_{cap}}$	Mean capacitor voltage	V
$\overline{V_{cap}}(t)$	Instantaneous mean capacitor voltage	V
V_{DS}	MOSFET drain-source voltage	V
X_T	Transformer leakage reactance	Ω
x^*	Reference	$-$
$\overline{x}$	Mean	$-$
$\tilde{x}$	Error	$-$
Z_L	Load impedance	Ω
Z_N	AC network impedance	Ω
δ_{ATB}	Tolerance band for average tolerance band capacitor balancing method	V
θ_{abc}	Phase angle, per phase	rad

Abstract

Name of University: The University of Manchester

Candidate's Name: Jack Andrews

Degree Title: Doctor of Philosophy

Thesis Title: Computational and Communication Architectures for Modular Multilevel Converter Construction

Date: November 2022

Renewable sources of electricity generation currently provide 47 % of the UK's total installed capacity. Offshore wind accounts for 13 % of the total and is set to increase significantly over the next decade, driven by the UK Government's target to reach net zero by the year 2050. An increasing number of offshore wind farms will use voltage source converter (VSC) high voltage direct current (HVDC) transmission schemes to transfer power to the onshore electricity grid. VSC-HVDC transmission is also being used to support the integration of renewables into future electricity grids. Given the rapid rollout of VSC-HVDC links, a thorough understanding of the technology is required.

At present, much of the research into VSC-HVDC in the public domain fails to capture the complexities of real-world control hardware, leading to inaccuracies in simulation models. Limited research has been carried out into the effect of controller implementation in software upon real-world processing delays in modular multilevel converters (MMC). The research presented in this thesis addresses these shortcomings by providing a detailed analysis of the delays in the capacitor balancing control (CBC) loop of an MMC.

Several sorting algorithms for use in the CBC loop have been implemented across a range of industrially representative control hardware and software platforms. The processing resource usage and execution delay have been measured to guide the algorithm selection process. A link between execution delay, choice of CBC method, and controller performance has been identified and is discussed. A simple method for incorporating this execution delay into a PSCAD/EMTDC simulation is then presented as a means of improving simulation model fidelity. A full suite of control software has been developed for a reduced-scale converter hardware prototype (CHP) MMC for future research into control loop delay and synchronisation. The implementation techniques developed are applicable to MMCs with a distributed control architecture in academia and industry.

Declaration

I hereby confirm that no portion of the work referred to in the thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright Statement

Acknowledgements

Firstly, I would like to express my gratitude to my two supervisors, Professor Peter R Green and Professor Mike Barnes, without whom this PhD would not have been possible. It has been a pleasure and a privilege to work alongside Peter and Mike for the past 3.75 years, and their insight and guidance has been invaluable throughout my research. I am thankful for the opportunities they have given me over the course of this project, and I look forward to continuing our collaboration and friendship in the years to come.

It would not have been possible to for me to undertake this research without financial support, therefore I wish to acknowledge the Engineering and Physical Sciences Research Council for their assistance in this area.

During this PhD, I have had many opportunities to work alongside numerous inspiring and talented colleagues. I would like to thank my fellow researchers in the Power and Energy Division for their stimulating discussions and technical assistance which helped guide this research and debug hardware problems on several occasions. A special thanks must go to Theo, Iñaki, and Paul for their support, not only in solving research-related problems, but also personally during the times when progress has felt slow.

I am forever thankful for the support of friends outside electrical engineering. In particular I would like to thank Josh, Ben, and Martin for their faithful friendship and support throughout the years during both good and bad times.

I must, of course, thank my family, especially my parents, to whom I owe a huge debt of gratitude. They have been unwavering in their support and encouragement throughout my entire life and have sacrificed so much to make me the person I am today. Thank you.

Above all, I thank God. I thank him for forgiving me and giving me life to the full, neither of which I deserve.

“*But God demonstrates his own love for us in this:*
While we were still sinners, Christ died for us.”

Romans 5:8 NIVUK

1 Introduction

1.1 Background

On 26th June 2019, the UK Government committed to a legally binding target to achieve net zero greenhouse gas emissions by 2050 [1]. An important part of the strategy to reach this target is increasing the proportion of renewable electricity generation in the UK's generation mix. Offshore wind will form a major contribution, with the UK Government setting a target of 40 GW of active generation by the year 2030, an increase from the previous target of 30 GW [2].

These targets are borne out in the allocation of 8 GW and 25 GW of generation capacity in the most recent leasing rounds of The Crown Estate and Crown Estate Scotland respectively [3, 4], increasing the total generating capacity of planned UK offshore wind projects to 86 GW as of March 2022 [5]. Going forwards, it is clear that offshore wind will play an increasingly important role in the generation of clean energy for the United Kingdom.

Worldwide, the picture is similar, with new installed offshore wind capacity showing a compound annual growth rate (CAGR) of 22 % in the decade up to the year 2020, and a forecast CAGR of nearly 30 % until the year 2025 [6]. At present, the growth in capacity is being led by China, the Netherlands, and Germany, however other countries are expected to play an increasing role in the next decade, as governments around the world raise their renewable energy ambitions [6].

1.1.1 Offshore Wind Farm Grid Connections

A key decision in the design of an offshore wind farm is the connection type used to transmit generated electricity to the onshore alternating current (AC) grid for distribution. This connection can be made using either high voltage alternating current (HVAC) or high voltage direct current (HVDC) along a subsea cable to shore. HVAC is a mature and well-understood technology, and as a result is often simpler and cheaper to implement than an equivalent HVDC scheme. HVDC transmission is more efficient for bulk power transfer, however can be significantly more expensive and complex to implement [7].

An important factor when choosing the connection type is the distance of the wind farm from the shore. Where HVAC is used to transfer power over a subsea cable, a large proportion of the cable's current carrying capacity is used to charge and discharge the cable capacitance every cycle. This reduces the amount of active and reactive power which can be transferred over the link, reducing its efficiency, and may require additional reactive power compensation, which increases costs [7]. Conversely, in a HVDC transmission system, once the cable capacitance is charged, almost the full current carrying capacity of the cable can be used for active power transfer.

Additional considerations include the subsea cable cost per kilometre, which is slightly higher for HVAC transmission, and the cost of the terminal substations, which are higher for HVDC transmission. As a result, a break-even point exists, based upon total system cost and transmission distance. For subsea transmission links such as those used for offshore wind farms, the break-even distance is typically in the range of 50-100 km, above which HVDC becomes more favourable [8]. This relationship is shown in Figure 1.1. The equivalent distance for HVDC using overhead lines is typically 300-800 km [8].

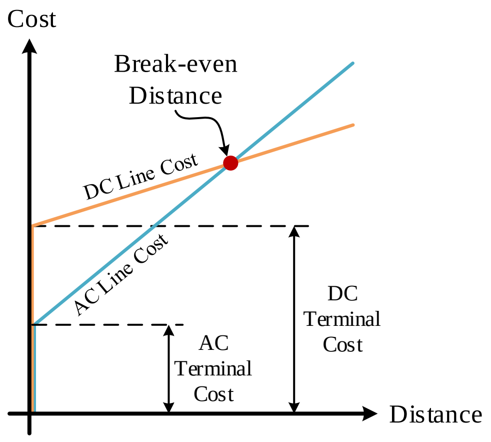


Figure 1.1: Subsea HVAC vs. HVDC transmission scheme break-even distance

In the UK, a number of consented Round 3 offshore wind farm developments will use HVDC for power transfer to shore, operating at distances between 75-215 km from shore [9]. Looking ahead to future projects, the six preferred sites selected by The Crown Estate in the Round 4 leasing process are at distances ranging from approximately 40-120 km from shore [10]. In the recent ScotWind leasing process, Crown Estate Scotland selected 17 projects with distances ranging from 5-80 km [11, 12]. At the time of writing, these projects are in early-stage planning and the transmission scheme (HVAC or HVDC) has not been decided.

There is a clear trajectory towards increasing the number of HVDC-connected offshore wind farms. In July 2020, the UK Government launched the Offshore Transmission Network Review (OTNR) [13], which aims to address the barriers to increasing offshore wind capacity. Central to this is the development of a strategy to coordinate the connection of offshore wind farms to the onshore grid. This may require the sharing of transmission assets by wind farm operators and construction of multi-purpose interconnectors (MPI) to connect neighbouring wind farms and countries using long-distance subsea cables. These requirements can be met by HVDC transmission links.

1.1.2 Renewable Electricity Generation Integration Challenges

Increasing the percentage of renewables in the generation mix, including other sources such as solar photo-voltaic (PV), hydroelectric, and tidal, brings with it additional challenges. Chief among these is maximising the utilisation of renewable generation when it is available, thereby avoiding costly curtailment fees and reducing overall dependence upon non-renewable sources of generation. The average curtailed GB wind generation across 2020-2021 was 2.9 TWh; enough energy to power 800,000 households [14]. In 2020, wind generation curtailment cost National Grid Electricity System Operator (ESO) £282 million in payments to wind farm operators, representing an additional cost of around £10 to each household [15].

Insufficient capacity on the existing transmission network has been identified as a primary cause of curtailment [14, 15]. During periods of high wind generation in the UK, a large amount of power is exported from generation sites in Scotland to demand centres in England, putting strain upon the existing transmission network. Without the installation of additional transmission capacity, the need to curtail generation will only be exacerbated by the planned connection of a further 30 GW of offshore wind generation by 2030.

Maximising the utilisation of renewable generation is dependent upon the ability to transfer large amounts of power from generation sites to demand centres. Within a country, these are often far apart and reinforcement of the grid using overhead AC transmission lines may be undesirable due to visual impact or planning concerns. In addition, interconnections between countries allow exporting of power during periods of excess renewable generation and vice-versa during times of low generation.

HVDC transmission links can satisfy both requirements and are a key enabler in the integration of renewable generation into the future electricity grid. Within a synchronous AC grid, subsea HVDC interconnectors can be used to strengthen the existing grid and increase power transfer capacity. Where interconnection of two asynchronous AC grids is required, HVDC transmission allows efficient, bi-directional bulk power transfer whilst maintaining a degree of isolation between the interconnected grids.

1.1.3 Voltage Source Converter HVDC

Two main types of converter technology are available commercially for HVDC transmission systems: current source converter (CSC) and voltage source converter (VSC). VSC-HVDC is better suited than CSC-HVDC for offshore wind farm grid connections and subsea interconnectors for several reasons. These include independent control of active and reactive power, and the ability to change power flow direction by reversing the direction of current flow, rather than the voltage polarity on the HVDC link [7]. This allows for easier integration into multi-terminal HVDC (MT-HVDC) schemes. Furthermore, VSC-HVDC can operate with a weak or inactive AC grid at one end of the transmission link, a critical requirement for offshore wind farm grid connections.

The first commercial VSC-HVDC scheme was the 3 MW Hellsjön-Grängesberg HVDC test system in Sweden, which was brought online in 1997 and used a two-level VSC [16]. All VSC-HVDC schemes used two- or three-level VSCs until 2010, when the modular multilevel converter (MMC) topology was used for the first time by Siemens on the Trans Bay Cable project in the United States [17]. MMC-based VSCs offer several advantages when compared to two- and three-level VSCs, including lower converter losses and a reduction in AC side harmonic content. The reduction in harmonic content reduces or eliminates the need for output filtering, leading to lower costs and a smaller converter footprint, which is desirable for VSCs installed on offshore platforms [18]. VSC-HVDC is currently employed in at least 51 HVDC schemes around the world, with approximately a further 60 projects scheduled to reach operational status in the next decade [19].

The internal operation of the MMC is more complex than previous two- and three-level VSCs and at present, much of the detailed understanding of the internal converter control is held by the manufacturers, who are unwilling to share for commercial reasons. In recent years, there has been a growing interest in more-open converter models, driven in part by electricity transmission system operators (TSO) who require accurate models which can be maintained for the lifetime of the converter to enable system studies to be carried out [20-22]. Despite this, more work is required to develop a better public domain understanding of the internal converter dynamics and implications of different control implementations.

Much of the public domain research and development fails to account for complexities in the underlying control hardware of the MMC. In particular, the processing delay introduced by the capacitor balancing control (CBC) loop in the MMC has been the subject of limited research in the public domain [23]. Failure to account for these delays can lead to unexpected consequences such as instability in control loops, premature aging of system components, or catastrophic component failure [24]. With the rapid roll-out of MMC-based HVDC links across the UK and globally, a better understanding of CBC loop dynamics in the presence of time delays is required, to identify and resolve sources of delay which could affect converter reliability. Furthermore, a more robust method to analyse the performance of the CBC loop in the presence of delays is required, so that effective comparisons can be made between different CBC methods.

The MMC topology has been chosen to investigate these issues since it has been the focus of much research and development in academia and industry over the last 15 years and is the dominant multilevel converter topology employed in industry at present. Furthermore, selecting the MMC topology for study allows use of a reduced-scale MMC converter hardware prototype (CHP) available in the department [25] for investigations on industrially-representative control hardware.

This research will focus on the internal control delays present in an MMC. In particular, the CBC loop will be studied and the implications of different control algorithm programming and implementation methods upon control delays will be investigated. The analysis techniques developed can be used to evaluate the performance of different CBC methods, with future extension to other time-critical control loops. The findings of this research can be used to inform the control algorithm implementation process for MMCs and improve the fidelity of electromagnetic transient (EMT) simulation models of MMCs.

1.2 Aims and Objectives

The aim of this thesis is to provide a detailed insight into internal control delays and controller implementation techniques for an MMC, to guide hardware and software development and improve simulation model fidelity. To achieve this broad aim, the following objectives have been identified:

1. Review and categorise CBC methods for an MMC and identify constraints on CBC loop execution delay.
2. Measure and compare the **resource usage** of a selection of sorting algorithms for CBC implemented on a range of industrially representative control hardware.
3. Measure and compare the **execution delay** of a selection of sorting algorithms for CBC implemented on a range of industrially representative control hardware.
4. Develop and validate a new suite of control software for the reduced-scale CHP which is suitable for investigating control loop delay and synchronisation.

1.3 Main Contributions

Thesis

The work contained in this thesis has made a number of contributions, predominantly in the area of MMC internal control delay measurement and modelling. The main contributions detailed in this thesis and associated conference [C] and journal [J] publications are summarised below:

- A thorough review of capacitor balancing control methods was carried out. The terminology around CBC methods has been clarified and a taxonomy of CBC methods has been developed as a basis for this work. This terminology and taxonomy developed for this work is also applicable more widely and is able to resolve some of the confusion around the literature on CBC.
- A range of sorting algorithms were implemented across three industrially representative control hardware platforms and three programming methods. The resource usage of each algorithm was measured and compared, providing a guide to sorting algorithm selection to meet control hardware resource constraints [J1, P1].
- The execution delay of the sorting algorithms was measured using representative capacitor voltage data as an input to the sorting algorithm. The results from this work can be used to guide sorting algorithm selection for time-constrained control loops [C1, P1] and to improve the fidelity of simulation models.

- A methodology to analyse the performance of the CBC loop in the presence of non-zero processing delays has been developed. The methodology has been applied to several simulation scenarios and has shown that delays above a certain time period degrade the performance of the CBC loop. The analysis method can also be applied to other control loops to evaluate their performance when processing or communication delays are present.
- A ground-up rewrite of the control software running on the reduced-scale MMC CHP was carried. The new control software resolves several issues with the previous control software, reduces processing resource usage, and allows configuration and measurement of internal control delays. The control implementation described in this thesis can be used to inform control software development for other prototype MMCs in academia and industry [J1].

Conference Papers [C]

1. J. Andrews, P. R. Green, M. Barnes, "A PSCAD Processor-in-the-loop System for Hardware Evaluation of Power Converter Control Algorithms", presented at the 10th IET International Conference on Power Electronics, Machines and Drives (PEMD), Online, Dec. 15-17, 2020, doi: 10.1049/icp.2021.1060.

Journal Papers [J]

1. T. Heath, M. Barnes, P. D. Judge, G. Chaffey, P. Clemow, T. C. Green, P. R. Green, J. Wylie, G. Konstantinou, S. Ceballos, J. Pou, M. M. Belhaouane, H. Zhang, X. Guillaud, J. Andrews, "Cascaded-and Modular-Multilevel Converter Laboratory Test System Options: A Review", *IEEE Access*, Vol. 9. pp. 44718-44737, Mar. 2021, doi: 10.1109/ACCESS.2021.3066261.

Additional

- Co-editor of the monthly "*VSC-HVDC Newsletter*" alongside M. Barnes. The newsletter focusses upon all aspects of VSC-HVDC technology including projects and novel technologies.
- Developed a public outreach demonstrator for the Holistic Operation and Maintenance for Energy from Offshore Wind Farms (HOME Offshore) project (Engineering and Physical Sciences Research Council grant EP/P009743/1).
- Developed a wireless underwater optical communication system for the Autonomous Aquatic Inspection and Intervention (A2I2) project (Innovate UK project number 104822).

Papers in Progress [P]

1. J. Andrews, T. Heath, P. R. Green, M. Barnes, “A Review and Performance Evaluation of Sorting Algorithms for Capacitor Balancing Control in Modular Multilevel Converters”, *Journal*

1.4 Thesis Structure

Chapter 2 – MMC Structure and Control

This chapter introduces the MMC topology as a foundation for the rest of the thesis. The circuit structure of a typical three-phase MMC employing half-bridge submodules is presented and analysed, followed by an explanation of the submodule capacitor voltage dynamics which will be recalled later in the thesis. An overview of the control structure of an industrial-scale HVDC MMC is then provided, focussing upon internal converter control loops.

Chapter 3 – Capacitor Balancing Control

A comprehensive review of capacitor balancing control methods is presented in this chapter. The terminology used to discuss the components of the CBC loop is defined, both as a basis for subsequent chapters and to clarify the often-confusing terminology used in the literature on CBC. A taxonomy of CBC methods is then developed based upon CBC loop sampling rate and sorting algorithm requirements. The results from the literature review have been used to guide selection of the CBC methods and sorting algorithms for study in Chapter 6.

Chapter 4 – Sorting Algorithms

This chapter provides an overview of sorting algorithms, their comparison metrics, and classification to ensure that the reader is sufficiently familiar with these for the discussion presented in Chapter 6. The comparison metrics are linked to the CBC loop execution delay and control hardware resource usage requirements.

Chapter 5 – Simulation Model Overview

In this chapter, the simulation model used to generate the synthetic capacitor voltage measurement data for the work in Chapter 6 is introduced and explained. The model used is an open-access PSCAD/EMTDC model developed by The National HVDC Centre and the University of Strathclyde [22]. The circuit structure of the model is introduced first, followed by the control structure and additional controllers which were not described in the MMC overview in Chapter 2. The operation of the custom PSCAD/EMTDC capacitor balancing control component developed for this work is also explained.

Chapter 6 – Sorting Algorithms for Capacitor Balancing Control

The results from the research into sorting algorithm execution delay and resource usage are presented in this chapter. The motivations for focussing upon the CBC loop are outlined first, followed by a description of the control hardware targets and programming languages used to implement the sorting algorithms. The execution delay results for each hardware target and programming method are then plotted and discussed, with reference to sorting algorithm structure and capacitor voltage dynamics. Finally, the field-programmable gate array (FPGA) logic resource usage of a sub-set of the sorting algorithms is measured, and the results are explained, identifying trade-offs between ease of implementation and logic resource usage.

Chapter 7 – Converter Hardware Prototype

This chapter provides an overview of the reduced-scale converter hardware prototype which has been used for hardware development in this research, with reference to [25]. The converter ratings and hardware structure are introduced first, followed by a description of the hardware modifications carried out during this work. Finally, a detailed description of the CHP control architecture is provided, as a basis for understanding the development carried out in Chapter 8.

Chapter 8 – Control Software Development

In this chapter, the ground-up rewrite of the CHP control software is described in detail. The limitations of the existing control software are identified first, and from these, a list of design objectives for the new control software are derived. The internal architecture of the CHP control system hardware is then described, with a particular focus upon data transfer, communication delays, and synchronisation between system components. The control software developed for the CHP is then documented, with descriptions of each component and how these operate together to ensure that the design objectives were met.

The CHP and the practical development work described in Chapters 7 and 8 forms the basis of a hardware platform for further study of the sorting algorithm execution delay in Chapters 3 to 6.

Chapter 9 – Conclusion and Future Work

This chapter provides a summary of the work presented in this thesis, and the main outcomes of the research are discussed. Opportunities for future research avenues which follow on from this work are also outlined.

Appendices and Supplementary Online Repository

Additional technical material, supplementary plots, and some of the source code developed during this project are included as appendices in this thesis. Other source code which is unsuitable for print is available for download in a supplementary online repository hosted by Mendeley Data. This data is accessible following [26], replicated here for ease of access.

[26] J. Andrews. *Supplementary Online Repository for Computational and Communication Architectures for Modular Multilevel Converter Construction*, Mendeley Data, 2022, doi: https://dx.doi.org/10.17632/fr2jrff9w3.1

2 MMC Structure and Control

This chapter provides an overview of the MMC topology as a foundation for subsequent chapters. The circuit structure of a three-phase HVDC MMC is presented, followed by a description of the half-bridge submodule (HB-SM) topology and the SM capacitor voltage dynamics. The circuit structure is then analysed and the control inputs to the voltage control loop of the converter are derived. A typical control structure representative of that used in a HVDC MMC is then introduced, with a focus on low-level converter control loops which are the subject of this research.

2.1 Circuit Structure

The circuit structure of a three-phase MMC is shown in Figure 2.1(a). Only one phase is shown for clarity; the other phases are identical. An MMC consists of three phase 'legs' connected in parallel across the DC link, which operates at a voltage, V_{dc}. Each phase leg comprises an upper and lower 'arm' which contain a number of submodules (SM), N_{SM}, connected in series with an arm inductor, L_{arm}. The arm inductor serves to limit the circulating current component of the arm current and the fault current rise rate [27]. A parasitic arm resistance, R_{arm}, is also shown in Figure 2.1(a); this represents the sum of the resistive losses in the SMs. The AC output voltage, $V_{ac(abc)}$, is taken from the mid-point of the upper and lower arms of each phase.

Each submodule contains two insulated-gate bipolar transistors (IGBT) and an energy storage capacitor and operates as a two-level converter; this is the half-bridge submodule topology and is shown in Figure 2.1(b). It is possible to use multiple IGBTs connected in series to increase the voltage across each SM; this is the cascaded two-level SM topology [28] and is not the subject of this explanation. Other SM topologies have been proposed in the literature and are implemented in industrial-scale MMCs [18, 29]; however, these are outside the scope of this work and will not be discussed here.

In the case of the HB-SM topology, the voltage at the SM terminals, V_{SM}, can be switched between 0 V and the voltage across the SM capacitor, V_{cap}, by controlling the two switches, S1 and S2. When S1 is closed and S2 is open, neglecting semiconductor device voltage drops, V_{SM} is equal to V_{cap} and the SM is 'inserted'. In this state, the SM capacitor, C_{SM}, will charge or discharge dependent upon the direction of the arm current, $I_{u,l}$. Conversely, when S1 is open and S2 is closed, V_{SM} is equal to 0 V and the SM is 'bypassed'. In this state, the SM capacitor voltage remains stable, neglecting self-discharge.

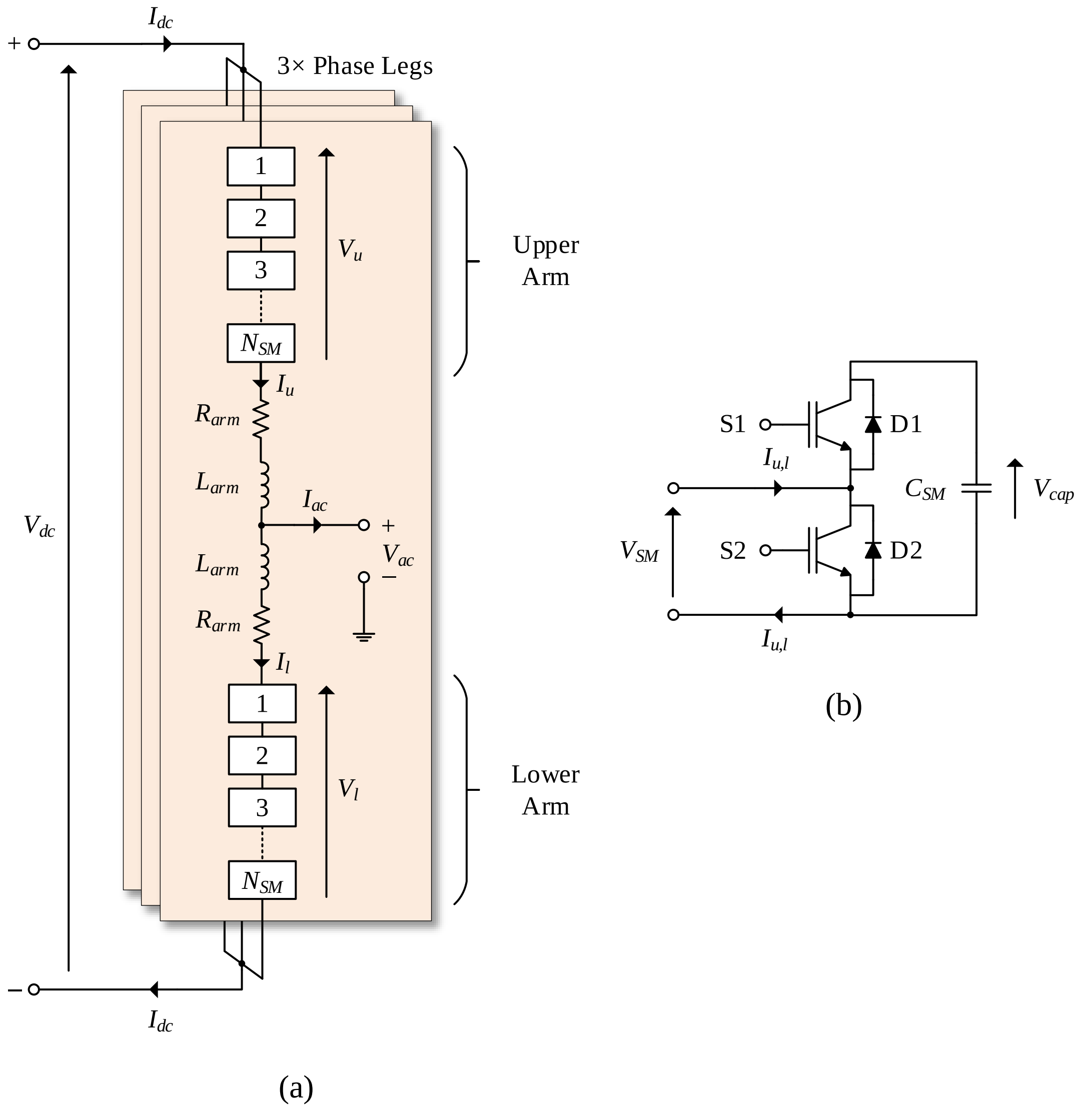


Figure 2.1: Circuit diagram of (a) modular multilevel converter, and (b) a half-bridge submodule

Two additional states exist. The 'blocked' state is formed when both switches are open and is typically only used during converter start-up to charge the SM capacitor via the freewheeling diode, D1. The state where both switches are closed is not permitted since this will short-circuit the SM capacitor and cause a large fault current to flow through S1 and S2, which may damage the switches or other SM components. The switch state combinations for a HB-SM are summarised in Table 2.1.

Arm Current	**S1**	**S2**	V_{SM}	C_{SM}	**State Name**
	0	0	V_{cap}	Charge	Blocked
$I_{u,l} > 0$	0	1	0 V	Stable	Bypassed
	1	0	V_{cap}	Charge	Inserted
	0	0	0 V	Stable	Blocked
$I_{u,l} < 0$	0	1	0 V	Stable	Bypassed
	1	0	V_{cap}	Discharge	Inserted
–	1	1	–	Short-circuit discharge	Not Permitted

Table 2.1: Half-bridge submodule switch states

By switching SMs in sequence, the string of SMs in each converter arm behaves as a controllable voltage source, with the smallest voltage step being equal to the SM capacitor voltage.

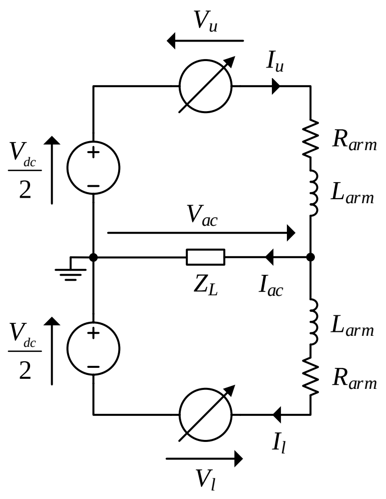


Figure 2.2: Modular multilevel converter equivalent circuit for a single phase

With reference to the equivalent circuit diagram for a single phase shown in Figure 2.2, two relationships for V_{ac} can be derived by applying Kirchoff's Voltage Law (KVL) around the upper and lower loops. These are shown in Equation 2.1 and Equation 2.2 for upper and lower loops, respectively.

$$V_{ac} = \frac{V_{dc}}{2} - V_u - R_{arm} I_u - L_{arm} \frac{dI_u}{dt} \qquad 2.1$$

$$V_{ac} = -\frac{V_{dc}}{2} + V_l + R_{arm} I_l - L_{arm} \frac{dI_l}{dt} \qquad 2.2$$

The upper and lower arm currents, $I_{u,l}$, are made up of three main components as shown in Equation 2.3 and Equation 2.4. The AC output current, I_{ac}, is divided equally between upper and lower arms. The remaining two terms are common to both arms and comprise of DC bus current, I_{dc}, shared across the three phases, and a double fundamental frequency circulating current, I_{circ}, which is caused by the voltage imbalance between phases [30].

$$I_u = \frac{I_{ac}}{2} + \frac{I_{dc}}{3} + I_{circ} \qquad 2.3$$

$$I_l = -\frac{I_{ac}}{2} + \frac{I_{dc}}{3} + I_{circ} \qquad 2.4$$

Substituting Equations 2.3 and 2.4 into Equations 2.1 and 2.2 respectively, then summing the resulting equations allows the AC output voltage to be expressed in terms of the upper and lower arm voltages, $V_{u,l}$, and the AC output current, as shown in Equation 2.5.

$$V_{ac} = \frac{V_l - V_u}{2} - \frac{R_{arm}}{2} I_{ac} - \frac{L_{arm}}{2}\frac{dI_{ac}}{dt} \qquad 2.5$$

A further quantity, the internal converter voltage, V_c, can be defined according to Equation 2.6 and is the voltage which drives the AC output current. Assuming a negligible voltage drop across the arm inductance and resistance, the internal converter voltage is approximately equal to the AC output voltage. The internal converter voltage is manipulated by the converter control loops to achieve the desired control objectives.

$$V_{ac} \approx V_c = \frac{V_l - V_u}{2} \qquad 2.6$$

2.1.1 Selection of Nominal Submodule Capacitor Voltage

The maximum AC output voltage is generated by bypassing all SMs in the upper arm and inserting all SMs in the lower arm. In this scenario, the AC output terminal is effectively directly connected to the positive pole of the DC bus (V_u = 0 V), and AC output voltage is equal to $V_{dc}/2$, ignoring the voltage drop across the arm inductance and resistance. The required lower arm voltage is calculated according to Equation 2.7.

$$V_u = 0, \quad V_{ac-max} = \frac{V_{dc}}{2} = \frac{V_l}{2} \quad \Rightarrow \quad V_l = V_{dc} \qquad 2.7$$

The minimum AC output voltage is generated by the opposite operation: inserting all SMs in the upper arm and bypassing all SMs in the lower arm. In this scenario, the lower arm voltage is equal to 0 V and the required upper arm voltage is calculated according to Equation 2.8.

$$V_l = 0, \quad V_{ac-max} = \frac{V_{dc}}{2} = \frac{V_u}{2} \quad \Rightarrow \quad V_u = V_{dc} \qquad 2.8$$

As can be seen from Equation 2.7 and Equation 2.8, the upper and lower arms must each be capable of generating a voltage equal to V_{dc}. This is achieved by ensuring that the sum of the SM capacitor voltages in the upper and lower arms, $V^{\Sigma}_{capu,l}$, are each equal to V_{dc}. As stated previously, the capacitors of inserted SMs are charged or discharged dependent upon the direction of the arm current, leading to a ripple on each SM capacitor voltage. The capacitor voltage ripples in an arm add up to produce a sum capacitor voltage ripple and as a result, only the long-term mean value of the sum SM capacitor voltage can equal V_{dc}. Ignoring the sum capacitor voltage ripple, the ideal nominal SM capacitor voltage, $V_{cap-nom}$, can be calculated according to Equation 2.9.

$$V_{cap-nom} = \overline{V_{cap}} = \frac{V^{\Sigma}_{capu,l}}{N_{SM}} = \frac{V_{dc}}{N_{SM}} \qquad 2.9$$

Generation of Arm Voltages

The voltage generated by an arm is equal to the number of inserted SMs in the arm (also known as the SM insertion index), $N_{onu,l}$, multiplied by the nominal SM capacitor voltage as shown in Equation 2.10. This forms the basis for controlling the arm voltage in an MMC.

$$V_u = N_{onu} \times V_{cap-nom} \qquad V_l = N_{onl} \times V_{cap-nom} \qquad 2.10$$

By appropriate control of the SM insertion index the output voltage magnitude and phase can be controlled independently. The number of output voltage levels, N_{level}, is equal to $N_{SM} + 1$.

Instantaneous Sum and Mean Submodule Capacitor Voltage

As stated previously, each arm exhibits a sum capacitor voltage ripple, leading to a corresponding ripple in the mean capacitor voltage – that is, $V^{\Sigma}_{capu,l}$ and $\overline{V_{cap}}$ vary with time. In this work, the time-varying version of $V^{\Sigma}_{capu,l}$ and $\overline{V_{cap}}$ are named the instantaneous sum capacitor voltage and instantaneous mean capacitor voltage, respectively, and are denoted by the (t) or $[k]$ suffix.

The instantaneous sum and mean capacitor voltages are introduced here since they are used by several capacitor balancing control (CBC) methods. Furthermore, the instantaneous versions of each variable can be used to quantify the effectiveness of CBC methods in ensuring that capacitor voltages remain balanced. These will be discussed further in Chapter 6.

2.2 Control Structure

Control of an MMC is typically achieved using several cascaded control loops. These control converter behaviour from power system-level dynamics such as active and reactive power transfer, down to low-level firing signals for the switches on individual submodules. The control objectives are dictated by the application scenario, for example, whether the converter is operating at the onshore or offshore end of an offshore wind farm HVDC link. A generalised representation of a cascaded control structure for an MMC is shown in Figure 2.3.

Active power (P), DC link voltage (V_{dc}), and the AC system fundamental frequency (f_0) can be controlled by varying the phase angle of the converter output voltage with respect to the connected AC network. Reactive power (Q), and the AC system voltage can be controlled by varying the magnitude of the converter output voltage. For a VSC-HVDC scheme connecting two active networks, one converter is configured to control the active power transfer or frequency, whilst the second converter controls the DC link voltage. Where a VSC-HVDC scheme is used to connect an offshore wind farm to an active network, the offshore converter controls the voltage and frequency of the offshore AC network, whilst the onshore converter controls the reactive power transfer and DC link voltage [31].

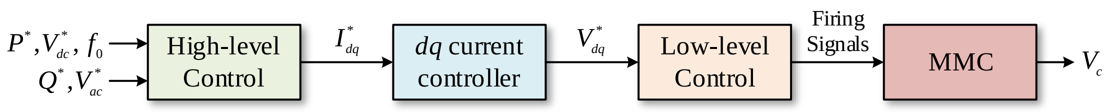


Figure 2.3: MMC cascaded control system overview

Since the power system-level quantities (P, Q, V_{ac}, V_{dc}, f_0) can be controlled by modifying the magnitude and phase angle of the converter output voltage, the voltage references from the high-level control loop can be fed directly to the low-level control loop (shown in yellow). This is termed direct control (not shown in Figure 2.3).

Alternatively, indirect (vector-based) control may be used, as shown in Figure 2.3. In this arrangement, a current control loop operating in the vector (dq) domain is inserted between the high- and low-level control loops. Vector output current control is described in this analysis since it is used in the MMC model used in this work and has several advantages when compared to direct control. These include inherent limiting of the current through the SMs and a faster response than direct control [18, 31].

The output of the vector output current controller is a dq domain voltage reference which is fed to the low-level control loop. Here, the voltage references are translated back into three-phase references and summed with voltage references from other controllers which control the internal dynamics of the MMC. The output of the low-level control is a set of firing signals which are applied to the SMs to generate the desired converter output voltage.

2.2.1 dq Current Control

Considering the simplified view of the connection of phase A to the AC network as shown in Figure 2.4, V_{ca} is the internal converter voltage as described previously, V_N is the network voltage, and Z_N is the network impedance. The voltage, V_{sa}, is the AC system voltage at the point of common coupling (PCC) and Z is the combined impedance between the converter arms and the PCC. This includes the arm inductance and resistance and the inductance and resistance of the interfacing transformer, where used.

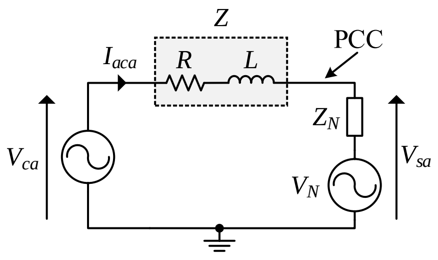


Figure 2.4: MMC connection to AC system, one phase shown

Applying KVL around the loop in Figure 2.4, the relationship between the internal converter voltage and AC system voltage is described by Equation 2.11.

$$V_{ca} - V_{sa} = RI_{aca} + L\frac{dI_{aca}}{dt} \tag{2.11}$$

Equation 2.11 can be reduced and represented in three-phase form as shown in Equation 2.12.

$$\begin{bmatrix} V_{csa} \\ V_{csb} \\ V_{csc} \end{bmatrix} = (R + Lp) \begin{bmatrix} I_{aca} \\ I_{acb} \\ I_{acc} \end{bmatrix} \qquad 2.12$$

where:

$$V_{cs(abc)} = V_{c(abc)} - V_{s(abc)} \qquad p = \frac{d}{dt} \qquad 2.13$$

Since vector current control is being used, Equation 2.12 must be translated from the three-phase abc reference frame to the dq domain. This is achieved using the Clarke-Park transformation. Equation 2.12 is first transformed into the $\alpha\beta0$ domain by applying the power-invariant Clarke transformation shown in Equation 2.14 to the three-phase voltages and currents, resulting in Equation 2.15.

$$\begin{bmatrix} \alpha \\ \beta \\ 0 \end{bmatrix} = \sqrt{\frac{2}{3}} \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} \qquad 2.14$$

$$\begin{bmatrix} V_\alpha \\ V_\beta \\ V_0 \end{bmatrix} = (R + Lp) \begin{bmatrix} I_\alpha \\ I_\beta \\ I_0 \end{bmatrix} \qquad 2.15$$

Assuming a balanced three-phase system, the $\alpha\beta0$ representation in Equation 2.15 can then be transformed again into the dq reference frame using the Park transformation in Equation 2.16.

$$\begin{bmatrix} d \\ q \end{bmatrix} = \begin{bmatrix} \sin(\theta) & -\cos(\theta) \\ \cos(\theta) & \sin(\theta) \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \qquad 2.16$$

Equation 2.12 can then be expressed in the dq reference frame according to Equation 2.17.

$$\begin{bmatrix} V_d \\ V_q \end{bmatrix} = R \begin{bmatrix} I_d \\ I_q \end{bmatrix} + Lp \begin{bmatrix} I_d \\ I_q \end{bmatrix} + \omega L \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} I_d \\ I_q \end{bmatrix} \qquad 2.17$$

Noting that $V_d = V_{cd} - V_{sd}$ and $V_q = V_{cq} - V_{sq}$, Equation 2.17 can be expanded to produce the two relationships shown in Equations 2.18 and 2.19.

$$V_{cd} - V_{sd} = RI_d + LpI_d - \omega LI_q \tag{2.18}$$

$$V_{cq} - V_{sq} = RI_q + LpI_q + \omega LI_d \tag{2.19}$$

The Laplace Transform ($\mathcal{L}$) with zero initial conditions can then be applied to Equations 2.18 and 2.19 to produce Equations 2.20 and 2.21.

$$V_{cd}(s) - V_{sd}(s) = RI_d(s) + LsI_d(s) - \omega LI_q(s) \tag{2.20}$$

$$V_{cq}(s) - V_{sq}(s) = RI_q(s) + LsI_q(s) + \omega LI_d(s) \tag{2.21}$$

Cross-coupling exists between the d and q components; the effects of this can be reduced using feedback nulling to de-couple the d and q components. Assuming that feedback nulling is used (which negates the ωL terms), Equations 2.20 and 2.21 can be rearranged to express I_d and I_q in terms of the d- and q-axis voltages as shown in Equations 2.22 and 2.23. This forms the basis of the MMC current control loop.

$$\frac{V_{cd}(s) - V_{sd}(s)}{Ls + R} = I_d \tag{2.22}$$

$$\frac{V_{cq}(s) - V_{sq}(s)}{Ls + R} = I_q \tag{2.23}$$

The MMC current control loop is shown in Figure 2.5. A proportional-integral (PI) feedback controller is used to control the d and q components of the output current independently. The d and q components of the AC system voltage (V_{sd} and V_{sq}) act as disturbances to the controller and can be nulled using feed-forward nulling [31] as shown in Figure 2.5.

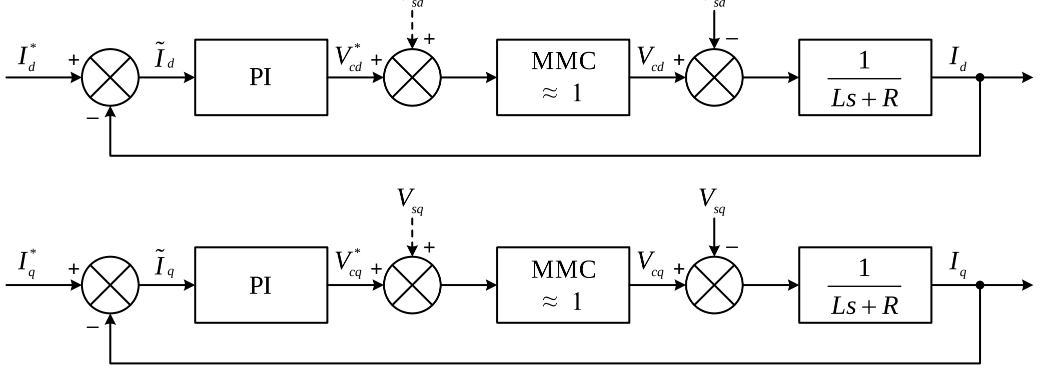


Figure 2.5: State feedback system block diagrams for MMC dq current control loop

In Figure 2.5, the MMC is represented as a unity gain block. This is a valid approximation assuming that the voltage output of the MMC accurately follows the input reference and that the MMC voltage control loop has a significantly higher bandwidth than the current controller. In order to decouple the dynamics of the voltage control and current control loops, the bandwidth of the inner voltage control loop should be four to ten times greater than the outer (enclosing) current control loop [25].

2.2.2 High-level (PQ) Control

The active and reactive power control loops are built around the current control loop according to Equations 2.24 and 2.25. The q-axis component of the AC system voltage (V_{sq}) is aligned with the MMC AC output voltage ($V_{ac(abc)}$) such that V_{sq} = 0, allowing Equations 2.24 and 2.25 to be simplified as shown on the right-hand side. As can be seen, the active and reactive power output can be controlled by adjusting the d- and q-axis current references respectively.

$$P = \frac{3}{2}(V_{sd}I_d + V_{sq}I_q) \qquad \Rightarrow \qquad P = \frac{3}{2}V_{sd}I_d \tag{2.24}$$

$$Q = \frac{3}{2}(V_{sq}I_d - V_{sd}I_q) \qquad \Rightarrow \qquad Q = \frac{3}{2}V_{sd}I_q \tag{2.25}$$

2.2.3 Low-level Control

The low-level control in an MMC is responsible for controlling the converter output voltage, in addition to internal converter dynamics specific to the MMC topology. The low-level control structure of an MMC is shown in Figure 2.6 and depicts the four main controllers: circulating current suppression control (CCSC), arm voltage control (AVC), nearest level control (NLC), and capacitor balancing control (CBC). An overview of each controller is provided in this section.

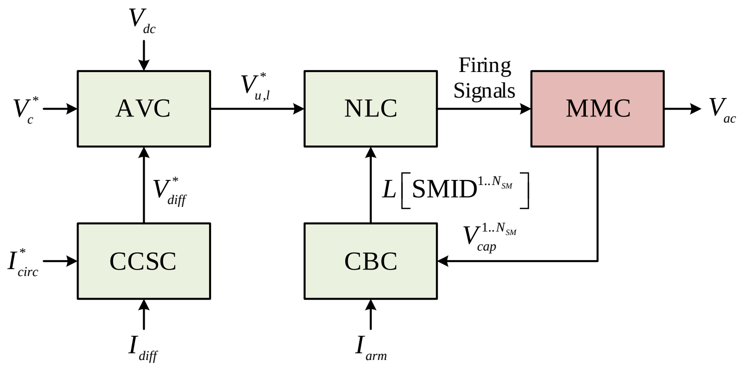


Figure 2.6: MMC low-level control structure

Circulating Current Suppression Control

As stated previously, an imbalance between the voltages across the phases of the converter exists due to the difference in stored energy per arm. This causes a circulating current to flow within the converter arms, which increases electrical losses and may necessitate the use of SM components with a higher current rating [30, 32].

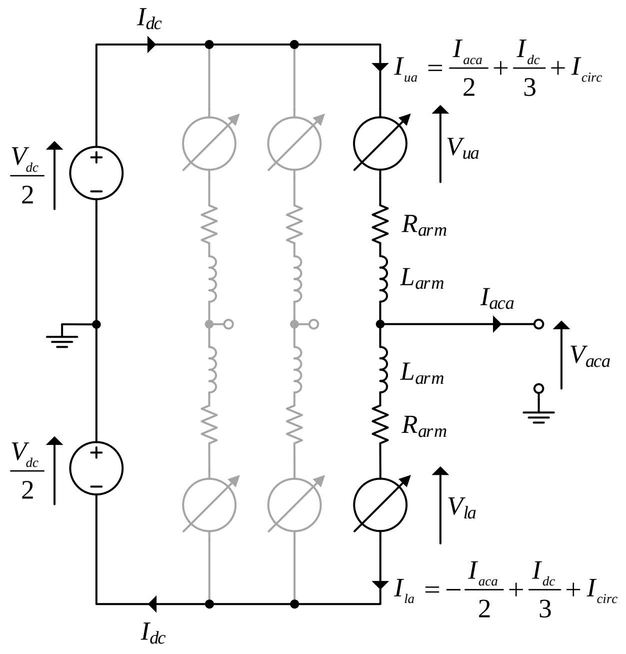


Figure 2.7: Equivalent circuit for a single phase of an MMC

Figure 2.7 shows converter phase A connected across the DC bus. Applying KVL around the loop, the voltage across the phase is derived according to Equation 2.26, where p is the first derivative function.

$$V_{dc} = V_{ua} + I_{ua}(R_{arm} + L_{arm}p) + I_{la}(R_{arm} + L_{arm}p) + V_{la} \qquad 2.26$$

The voltage drops across the arm inductor and resistor due to the AC output current component (I_{aca}) cancel each other out due to the current having opposing signs in each arm. The arm currents can therefore be replaced with the difference current, I_{diff}, which is common to both arms and is shown in Equation 2.27. By substituting I_{diff}, and rearranging, Equation 2.26 can be reduced to Equation 2.28.

$$I_{diff} = \frac{I_{dc}}{3} + I_{circ} \qquad 2.27$$

$$V_{dc} - \frac{V_{ua} + V_{la}}{2} = I_{diffa}(R_{arm} + L_{arm}p) = V_{diffa} \qquad 2.28$$

The right-hand side of Equation 2.28 is termed the difference voltage, V_{diff}, and is the voltage drop across a single arm caused by the difference current. Assuming a small parasitic arm resistance, the voltage drop across the arm resistance is negligible. The main contribution to V_{diff} is from the voltage developed across the arm inductor due to the circulating current, which is a negative sequence current at double the fundamental frequency [32]. As a result, the circulating current can be suppressed by controlling the difference voltage to be equal to zero.

The difference voltage can be controlled by manipulation of the upper and lower arm voltages as demonstrated previously in Equation 2.28. The right-hand side of Equation 2.28 can be expressed in matrix form for three phases as shown in Equation 2.29; this is the plant equation for the circulating current within an MMC.

$$\begin{bmatrix} V_{diffa} \\ V_{diffb} \\ V_{diffc} \end{bmatrix} = (R_{arm} + L_{arm}p) \begin{bmatrix} I_{diffa} \\ I_{diffb} \\ I_{diffc} \end{bmatrix} \qquad 2.29$$

A range of methods to control the circulating current have been proposed in the literature. Since Equation 2.29 is of the same form as the output current plant equation in Equation 2.12, the authors in [25, 30, 31, 33] implement CCSC in the dq domain using a PI controller with the d- and q- axis current references set to zero. Alternatively, the three phase difference voltages can be controlled directly using a proportional (P) or proportional-resonant (PR) controller as described in [34] and [18]. The simulation model used in this work as described in Chapter 5 uses a PR controller to control the circulating current. Circulating current control is not the focus of this work and will not be discussed further here.

Arm Voltage Control

The arm voltage controller is responsible for combining the voltage references from the output current controller and CCSC to produce independent upper and lower arm voltage references. It satisfies the simultaneous equations for the MMC arm voltages as shown in Equation 2.30 and 2.31, derived from Equation 2.6 and 2.28 respectively.

$$V_l - V_u = 2V_c \qquad 2.30$$

$$V_u + V_l = V_{dc} - 2V_{diff} \qquad 2.31$$

Solving Equations 2.30 and 2.31 for the upper and lower arm voltages produces two equations for the upper and lower arm voltage references as shown in Equations 2.32 and 2.33.

$$V_u^* = \frac{V_{dc}}{2} - V_c^* - V_{diff}^* \qquad 2.32$$

$$V_l^* = \frac{V_{dc}}{2} + V_c^* - V_{diff}^* \qquad 2.33$$

Nearest Level Control

The upper and lower arm voltage references are then passed to the modulation algorithm, which translates the voltage commands into firing signals for individual SM switches. In this analysis, nearest level control modulation is described since it is the preferred modulation algorithm for industrial-scale HVDC MMCs with several hundred SMs per arm [18]. This is due to a simpler implementation when compared to pulse-width modulation (PWM) schemes, which require generation and distribution of N_{SM} PWM carriers per converter arm.

The output of the NLC algorithm is a pair of upper and lower arm SM insertion indices, $N_{onu,l}$, as introduced in Section 2.1.1. The SM insertion indices are calculated from the arm voltage references and the nominal capacitor voltage $V_{cap-nom}$ according to Equation 2.34.

$$N_{onu} = \text{round}\left(\frac{V_u^*}{V_{cap-nom}}\right) \qquad N_{onl} = \text{round}\left(\frac{V_l^*}{V_{cap-nom}}\right) \qquad 2.34$$

The $\text{round}()$ operator returns the nearest integer to the result of the division operation. This is required since only integer numbers of SMs can be inserted to generate the required output voltage. Using the calculated value of N_{on}, the NLC loop then selects specific SMs to insert or bypass in conjunction with the capacitor balancing control loop.

Capacitor Balancing Control

When NLC modulation is used, the ability to meet the desired arm voltage reference is predicated on the SM capacitor voltages being balanced and maintained at the nominal capacitor voltage; this is evident from Equation 2.34. As explained in Section 2.1, SM capacitors charge or discharge dependent upon the SM state and the direction of the arm current, causing SM capacitor voltages to deviate from the nominal value. As a result, a secondary controller is required to ensure that SM capacitor voltages are maintained around the target value.

The capacitor balancing control loop ensures that SM capacitor voltages remain balanced by selecting specific SMs for insertion or bypassing by the NLC loop based upon the measured SM capacitor voltages. When triggered, the CBC loop measures the capacitor voltages of all SMs in an arm and places them into an unsorted array with a corresponding SM identifier (SMID). A sorting algorithm then sorts the SMID:V_{cap} value pairs in ascending or descending order, according to the direction of the arm current.

If the arm current is positive the capacitors of inserted SMs will charge, therefore the SMs with the lowest capacitor voltages should be prioritised for insertion. This is achieved by sorting the list in ascending order. Conversely, if the arm current is negative, the capacitors of inserted SMs will discharge, so the SMs with the highest capacitor voltages should be prioritised by sorting the list in descending order. The NLC loop then selects N_{on} submodule identifiers for insertion starting at the beginning of the sorted array.

The capacitor balancing control loop is a key focus of this work and is described in more detail in subsequent chapters, therefore it will not be discussed further here.

2.3 Summary

This chapter has provided an overview of the theory of operation and control of a typical HVDC MMC as a foundation for the subsequent chapters in this thesis. The circuit structure was presented and the equations which govern the converter output voltage dynamics were derived. The control structure of the MMC was then described, with a focus upon cascaded control and the low-level control loops. The control structure and controller implementations described in this chapter are one common method of controlling MMC dynamics, however a wide range of methods have been proposed in the literature. A detailed review of MMC control methods can be found in [35].

3 Capacitor Balancing Control

The CBC loop (also referred to as voltage balancing control and SM energy balancing control) ensures that the SM capacitor voltages are maintained about a nominal set point and within the safe operating limits of the SM. A sorting algorithm is used in many CBC methods as part of the balancing process.

This chapter provides a review of capacitor balancing control methods presented in the literature. The terminology surrounding CBC is defined and clarified first, as a basis for the literature review and later chapters. A selection of capacitor balancing control methods are then reviewed and two taxonomies are developed based upon the requirement for a sorting algorithm and the CBC method sampling period. The CBC methods discussed are then categorised according to this taxonomy and three methods are selected for further analysis in the work presented in Chapter 6.

Capacitor balancing can be implemented either pre-modulation or post-modulation [36]. In the pre-modulation scheme, an additional term is added to the per-SM or per-arm voltage reference to balance the SM capacitor voltages. This is commonly referred to as closed-loop capacitor balancing.

In the post-modulation scheme, the modulation algorithm determines the number of SMs to insert, whilst the CBC loop determines the specific SMs which should be inserted, bypassed, or pulse-width modulated to maintain capacitor voltage balancing. This work focusses upon post-modulation CBC methods.

Capacitor balancing may be implemented locally at the SM level, or at the arm level in a distributed control unit located ‘centrally’ in the main converter control cubicle. This work focusses upon CBC methods which operate at the arm level and assumes that the CBC loop has access to the capacitor voltage measurements of all SMs in an arm.

3.1 Terminology

The relationship between modulation algorithm, CBC method, sorting algorithm, and their inputs and outputs is shown in Figure 3.1. Much of the published literature on CBC methods and sorting algorithms uses these terms interchangeably or fails to make a distinction between the CBC method and the sorting algorithm; the terminology in use is clarified and explained in the following sections.

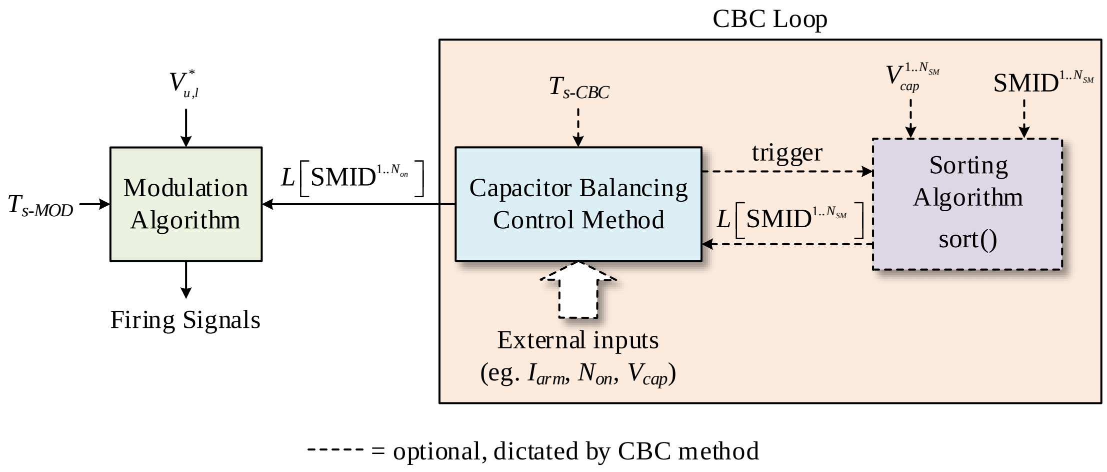

CBC Loop
$V_{u,l}^{*}$
$T_{s\text{-}MOD}$
Modulation Algorithm
Firing Signals
$L\left[\text{SMID}^{1..N_{on}}\right]$
$T_{s\text{-}CBC}$
Capacitor Balancing Control Method
trigger
$L\left[\text{SMID}^{1..N_{SM}}\right]$
$V_{cap}^{1..N_{SM}}$
$\text{SMID}^{1..N_{SM}}$
Sorting Algorithm
sort()
External inputs
(eg. I_{arm}, N_{on}, V_{cap})
----- = optional, dictated by CBC method

Figure 3.1: Block diagram showing relationship between modulation algorithm, CBC method, and sorting algorithm

3.1.1 Modulation Algorithm

The modulation algorithm is responsible for generating the firing signals for the SMs. It receives the arm voltage reference ($V_{u,l}^{*}$) from the arm voltage control loop and a list of SM identifiers ($L[\text{SMID}^{1..N_{on}}]$) from the CBC loop corresponding to the specific SMs to insert. The modulation algorithm sampling period (T_{s-MOD}) is decoupled from the CBC method sampling period (T_{s-CBC}), that is, the two loops can execute independently.

A range of modulation algorithms have been proposed in the literature published to-date. For the purposes of this work, nearest level control modulation has been assumed where necessary, since it is favoured for use in industrial scale converters due to ease of implementation with several hundred SMs and low switching frequency when compared to PWM-based algorithms [18].

3.1.2 Capacitor Balancing Control Method

The capacitor balancing control method is at the core of the CBC loop. In this work, the term ‘method’ is used instead of algorithm to further reinforce the distinction between the CBC method, modulation algorithm, and sorting algorithm. The CBC method implements two key functions:

- Evaluates the conditions which will trigger an update of the output list ($L[\text{SMID}^{1..N_{on}}]$) and thus switching of SMs for balancing purposes.
- Determines the switching actions performed to balance SM capacitor voltages, such as which SMs are targeted for sorting and balancing.

Dependent upon the chosen CBC method, balancing may be triggered with a fixed sampling period (T_{s-CBC}) or conditionally, using external inputs, as shown in Figure 3.1. Where capacitor balancing is triggered conditionally on external inputs, the sampling period may vary from one sampling instant to the next and is more accurately described as an effective sampling period.

When triggered, the CBC method may require the use of a sorting algorithm to generate a sorted list of all SM capacitor voltages and SM identifiers, or a sorted list of a subset of SMs determined by the CBC method. CBC methods which require the use of a sorting algorithm are typically referred to as ‘sorting-based’ balancing methods in the literature and are the focus of this chapter.

3.1.3 Sorting Algorithm

The sorting algorithm is triggered by the CBC method and takes a set of SM capacitor voltage measurements acquired from the SMs ($V_{cap}^{1..N_{SM}}$) and a corresponding set of SM identifiers ($\text{SMID}^{1..N_{SM}}$) as inputs. The algorithm then sorts the $\text{SMID}: V_{cap}$ value pair in ascending or descending order according to the SM capacitor voltage value. The operation of the sorting algorithm is explained in more detail in Chapter 4.

3.2 Review of Capacitor Balancing Control Methods

Capacitor balancing control methods can be classified based upon different parameters such as: switching frequency, capacitor balancing performance, or choice of sorting algorithm [18, 35, 37]. Whilst these classifications are valid, they fail to capture the distinctions between CBC methods which are necessary for understanding the research presented in this chapter. As a result, two new taxonomies for classifying CBC methods have been developed based upon sorting algorithm requirement and CBC loop sampling period. These taxonomies are introduced in this section and the existing literature on CBC methods is reviewed and placed into these categories.

3.2.1 Sorting Algorithm Requirement

As stated previously, some CBC methods require the use of a sorting algorithm to generate a sorted list of SM identifiers each time the CBC loop is triggered. Other CBC methods do not require the use of a sorting algorithm and between these categories further distinctions can be made. This leads to a taxonomy of CBC methods based upon the requirement for a sorting algorithm, this is shown in Figure 3.2.

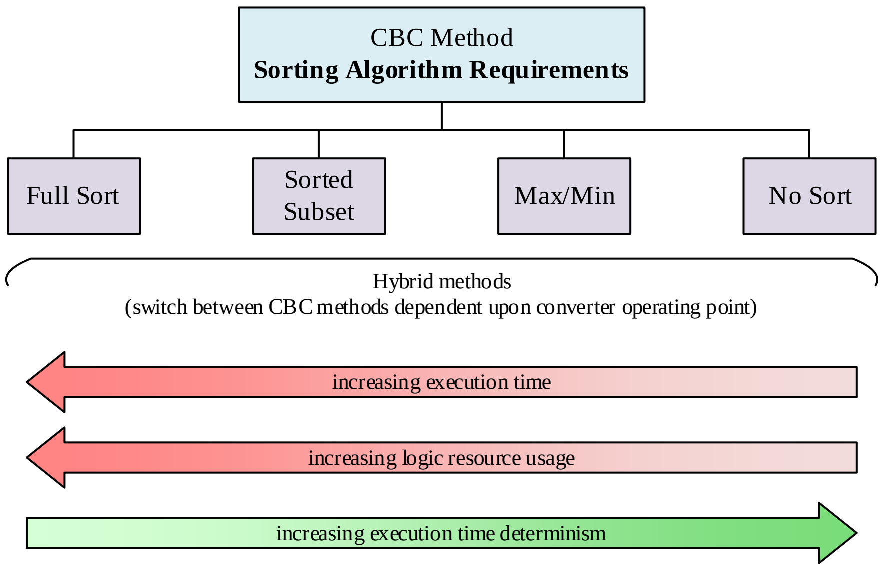

CBC Method
Sorting Algorithm Requirements
Full Sort
Sorted Subset
Max/Min
No Sort
Hybrid methods
(switch between CBC methods dependent upon converter operating point)
increasing execution time
increasing logic resource usage
increasing execution time determinism

Figure 3.2: CBC method taxonomy based upon sorting algorithm requirements

The four categories presented in Figure 3.2 will be explained in further detail in subsequent sections. A fifth category containing 'hybrid' methods can also be defined for CBC methods which switch between methods in one of the four primary categories dependent upon the converter operating point.

Typically, the execution time of the overall CBC loop will increase when moving from CBC methods which do not require use of a sorting algorithm ('no sort') to those which require a fully sorted list of SM identifiers ('full sort'). This is due to the use of a sorting algorithm which executes at each firing of the CBC loop. In the case of maximum/minimum ('max/min') or no sort CBC methods, the processing operations carried out by the CBC method are usually less complex and will execute in a shorter time period than a sorting algorithm. Furthermore, the field-programmable gate array (FPGA) logic resource usage of the overall CBC loop will typically increase for full sort methods, due to the additional logic resources required by the sorting algorithm.

When moving from full sort CBC methods to no sort methods, the CBC loop execution time will typically become more deterministic. When compared to the classical sorting algorithms commonly used in sorting-based CBC methods, the processing operations performed in max/min and no sort methods have fewer nested loops and conditional branches. These structures are partly responsible for the non-deterministic execution time exhibited by many classical sorting algorithms. The work carried out in this chapter focuses on CBC methods which require a full sort, since these represent the worst-case execution time and logic resource usage for the overall CBC loop.

Full Sort

The majority of CBC methods which have been published in the literature to date require a fully sorted list of SM identifiers each time the CBC loop is triggered. The original proposal for the MMC by Lesnicar and Marquardt in [38] relied upon a fully sorted list to select specific SMs to inserted or bypass to ensure balancing. This CBC method is also used in [39] and is termed ‘slow-rate capacitor balancing’ or the ‘classical approach’ in [40].

Three tolerance band-based CBC methods have been presented in the literature; these aim to reduce SM switching instances for balancing purposes and the associated switching losses. The average tolerance band (ATB) and cell tolerance band (CTB) methods are presented in [41-43]. An additional tolerance band method named cell tolerance band optimised (CTBoptimised) is introduced by the same authors in [43, 44]. Each of these methods require a fully sorted list of SM identifiers whenever the CBC loop is triggered.

The authors in [39] present a fundamental frequency capacitor voltage balancing method which also requires a fully sorted list of SM identifiers upon each CBC loop iteration. In this method the SM switching frequency for balancing purposes is reduced by only triggering the CBC loop when the AC side voltage level is $\pm V_{dc}/2$.

In addition to the CBC methods outlined previously, two papers have proposed predictive methods which also rely upon a full sort of either the SM capacitor voltages or the predicted capacitor voltage errors [39, 45]. Furthermore, many of the post-modulation PWM-based CBC methods also require a fully sorted list of SM identifiers for operation [46-48].

Sorted Subset

Several CBC methods have been proposed which require the sorting of only a subset of SM capacitor voltages – these are termed ‘sorted subset’ methods for the purpose of this work. The SMs targeted for sorting are first determined by the CBC method then passed to the sorting algorithm. A sorting algorithm is still required by all these methods, however by definition of the sorted subset category, the number of elements to sort (n) will fall in the range: $1 \leq n \leq N_{SM}$.

The authors in [33] present a reduced switching frequency (RSF) CBC method which only requires sorting of a subset of submodules dependent upon the value of ΔN_{on}. This method can only be used with phase-shifted carrier PWM (PSC-PWM) and is unsuitable for use with NLC modulation due to an insufficient number of switching instances.

A number of CBC methods have been proposed which search for only the N_{on} SMs with the highest or lowest capacitor voltages, at which point the sorting process is halted [49-52]. Alternatively, the authors in [53] propose a pre-selection method for SMs of interest to reduce the number of SM capacitor voltages to sort. These methods can also be classified as sorted subset CBC methods.

Maximum/Minimum

Several authors have proposed CBC methods which identify only the SMs with the maximum/minimum capacitor voltages for switching for balancing purposes. Unlike the classical sorting algorithms commonly used in full sort CBC methods, the execution time for searching for the maximum/minimum SM capacitor voltages is more deterministic and scales linearly as N is increased.

The authors in [54] present a max/min CBC method using a voltage-controlled oscillator (VCO) on each SM to measure the capacitor voltage. The VCO signals from each SM are then fed into digital logic which counts the VCO pulses and selects the SMs with the max/min capacitor voltages for balancing in a method termed ‘the tortoise and the hare’.

Another max/min CBC method named ‘selective virtual loop mapping’ is presented in [55] and is implemented alongside phase-disposition PWM (PD-PWM). The same method is also used by the same authors in [56] and is instead called ‘selective bias loop mapping’. This method has the advantage of a reduction in execution time due to parallelisation of the max/min search. Other max/min CBC methods are presented in [57] and [58].

No Sort

Capacitor balancing can also be implemented without using a sorting algorithm or max/min search; these methods are titled no sort CBC methods for the purpose of this work. Several closed loop CBC methods have been proposed. These can operate at either the SM-local level by modifying the per-SM voltage reference as in [59], or at the arm level by monitoring the arm energy and adjusting the arm voltage reference to maintain balancing, as described in [37] and [60]. Other CBC methods which do not require a sort also include: cell tolerance band with sequence reversing (CTBsequence) [42, 43] or model predictive control methods [61, 62].

Hybrid Methods

A number of ‘hybrid’ CBC methods have also been proposed in the literature. These methods typically implement one CBC method during steady state operation, usually with the aim to reduce SM switching frequency, then switch to another CBC method during transients to ensure that capacitor balancing is maintained. The requirement for a sorting algorithm is dictated by the CBC methods chosen as part of overall hybrid method.

Hybrid methods which switch between classical full sort balancing and the RSF method are proposed in [40]. The conditions associated with triggering CTB and ATB as presented in [42] are used to select whether the classical full sort balancing method or the RSF method is used, typically during transients and steady state respectively. A similar hybrid method is described in [63], which switches between CTBoptimised in steady state and ATB during transients.

The authors in [39] present a hybrid method which switches between the classical full sort method and a predictive method based upon a comparison of the magnitude of the predicted SM capacitor voltage error and a fixed tolerance.

3.2.2 Sampling Period

As outlined at the beginning of this section, CBC methods can also be categorised based upon their sampling period, as shown in Figure 3.3. Classifying CBC methods in this way is required for this work, since the CBC loop sampling period dictates the time window available for the sorting algorithm to execute for sorting-based CBC methods.

CBC methods can be classified under one of two main categories: fixed sampling period (periodic), or variable sampling period (aperiodic). In the case of periodic CBC methods, the CBC loop may be triggered by a sample clock dedicated to the CBC loop (local), or by an external sample clock (external). Due to the periodicity of the sample clock, the sorting algorithm used by the CBC method has a pre-determined, fixed time period during which it must finish executing.

In the case of aperiodic CBC methods, the CBC loop may be triggered according to conditions on measurements which are either ‘local’ or ‘external’ to the CBC loop. Unlike periodic methods, the time period available for a sorting algorithm to execute is variable and may vary from one loop iteration to the next.

In this classification, the term ‘local’ is used to refer to measurements which are required primarily by the CBC loop itself, namely SM capacitor voltage. ‘External’ measurements are those which are primarily associated with other control loops, but which may be used to trigger the CBC loop, for example, the SM insertion index (N_{on}). Whilst the local and external categories are shown in Figure 3.3, these are less well-defined than the main periodic and aperiodic categories, however, are shown for completeness.

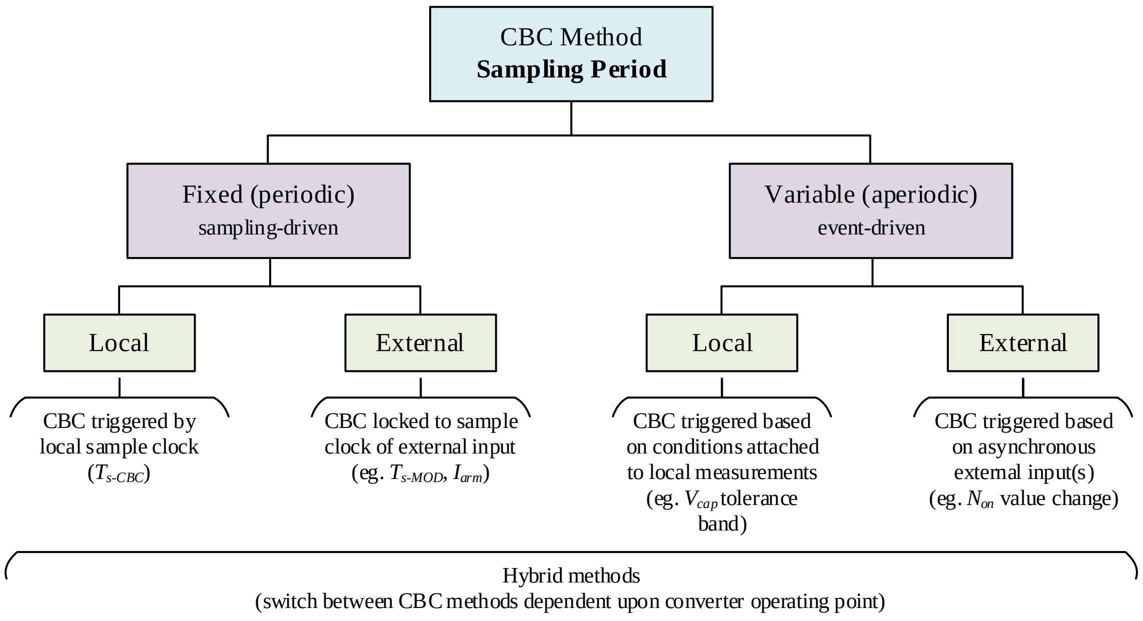

CBC Method
Sampling Period
Fixed (periodic)
sampling-driven
Variable (aperiodic)
event-driven
Local
External
Local
External
CBC triggered by local sample clock ($T_{s\text{-}CBC}$)
CBC locked to sample clock of external input (eg. $T_{s\text{-}MOD}$, I_{arm})
CBC triggered based on conditions attached to local measurements (eg. V_{cap} tolerance band)
CBC triggered based on asynchronous external input(s) (eg. N_{on} value change)
Hybrid methods
(switch between CBC methods dependent upon converter operating point)

Figure 3.3: CBC method taxonomy based upon CBC loop sampling period properties

Periodic CBC methods can also be said to be sampling-driven, whilst aperiodic CBC methods can be said to be event-driven. Event-driven CBC methods are triggered dependent upon a condition being met, such as a SM capacitor voltage exceeding the tolerance band or a change in an input variable. As with the taxonomy based upon sorting algorithm requirement, a fourth category can also be defined for hybrid CBC methods which switch between methods dependent upon the converter operating point. Hybrid methods will typically have a variable sampling period.

Periodic: Local

Several CBC methods have been proposed which are triggered periodically by a sample clock local to the CBC loop itself. In the classical approach originally proposed by Lesnicar and Marquardt in [38] and used by the authors in [39, 40], the CBC loop is triggered at a fixed sampling period which is chosen to ensure SM capacitor voltages remain balanced across all converter operating points. The hybrid CBC method proposed in [39] is also triggered at a fixed sampling period.

Periodic: External

Other CBC methods have been presented in the literature which are triggered by a periodic sample clock which is primarily used to drive a control loop external to the CBC loop itself. In this configuration, the CBC loop sampling period is locked to the sampling period of the external control loop. Typical examples of CBC methods in this category are those which are triggered at each cycle of the carrier waveform in the case of PWM-based modulation algorithms [46-48].

Aperiodic: Local

Aperiodic CBC methods have a variable sampling period since these methods are triggered conditionally on measurements such as SM capacitor voltage. For example, the CTB and ATB methods introduced in [41-43] are only triggered when a SM capacitor voltage exceeds a maximum or minimum value, or a tolerance band around the average SM capacitor voltage respectively. As a result, during some control cycles the CBC loop is not triggered and a sorted list of SM identifiers is not required. During transients however, the CBC loop may be triggered on consecutive control cycles, leading to a shorter time window during which the sorting algorithm must execute. The CTBoptimised method introduced in [43, 44] is another example of an aperiodic CBC method; in this case, the CBC loop is triggered on zero crossings of the arm current.

Aperiodic: External

Several aperiodic CBC methods have been proposed which rely upon external measurements which are not typically associated with the CBC loop. For example, the RSF method proposed in [33] and the predictive strategy presented in [45] are both triggered by changes in the SM insertion index (i.e. when $N_{on}[k] \neq N_{on}[k-1]$). Alternatively, the fundamental frequency balancing method in [39] is triggered only when the AC side voltage level is $\pm V_{dc}/2$. Whilst it is possible that under ideal steady state operating conditions the parameters used to trigger these CBC methods will be periodic, this is not necessarily the case during transients, therefore these methods are classified as aperiodic.

3.3 Summary

This chapter has provided a detailed review of CBC methods presented in the literature and developed two taxonomies for classifying CBC methods which are relevant to the research presented in Chapter 6. Whilst there are no widely-accepted standard CBC methodologies used in industry made available in the public domain, the reviewed methods arise from both academic and leading industrial sources [43, 64]. In addition, the terminology specific to CBC has been clarified and a clear distinction made between the functions of the modulation algorithm, CBC method, and sorting algorithm.

4 Sorting Algorithms

This chapter provides an overview of sorting algorithms as a basis for subsequent chapters in this thesis. The $\mathrm{sort}()$ operation is formally defined first, followed by a review of the comparison metrics which are commonly used to compare sorting algorithm performance. These comparison metrics will be returned to in later chapters. Finally, a special case of sorting algorithm known as the sorting network is introduced and explained. Sorting networks are included since they exhibit desirable execution time characteristics for time-constrained real-time control loops.

In sorting-based CBC methods, a sorting algorithm is used to generate an ordered list of SM identifiers using the corresponding SM capacitor voltages each time the CBC loop is triggered. Formally, for an input set, A, comprising n elements ($A = \{a_1, a_2, \dots, a_n\}$) the sorting algorithm performs the $\mathrm{sort}()$ operation to produce an ordered output set, A':

$$\mathrm{sort}(A) \rightarrow A' = \{a'_1, a'_2, \dots, a'_n\} \tag{4.1}$$

where:

$a'_1 \geq a'_2 \geq \cdots \geq a'_n$ for a sort in ascending order

$a'_1 \leq a'_2 \leq \cdots \leq a'_n$ for a sort in descending order

A' is a permutation of A, that is, it includes all the elements in A.

In the case of the CBC loop, only the list of ordered SM identifiers are passed to the CBC method for further processing. As stated in Section 3.2.1, in some CBC methods the sorting algorithm may only be required to sort a subset of SM capacitor voltage measurements.

4.1 Comparison Metrics

To provide an objective comparison between the performance of different sorting algorithms, two key metrics are commonly used [65]. These are termed ‘time complexity’ and ‘space complexity’ and are defined as follows:

- Time complexity: describes how the execution time of a sorting algorithm scales as the number of inputs to be sorted, n, is increased.
- Space complexity: describes how the resource usage of a sorting algorithm scales as n is increased.

These terms are described in more detail in the following sections.

4.1.1 Bachmann-Landau (Big-*O*) Notation

In the theoretical domain, a subset of the Bachmann-Landau notations, known in computer science as 'big-*O*' notation, is used to describe the asymptotic upper bound on the growth rate of a function as the argument tends to infinity [65]:

$$f(n) = O\big(g(n)\big) \text{ as } n \to \infty \tag{4.2}$$

In the case of sorting algorithms, big-*O* notation is used to compare the growth rate of algorithm time and space complexity as the number of elements in the input set is increased. For sorting algorithms used in CBC, n is equal to the number of SMs per arm (N_{SM}). Some common growth functions are plotted in Figure 4.1 for n = 1 to 500 for comparison purposes.

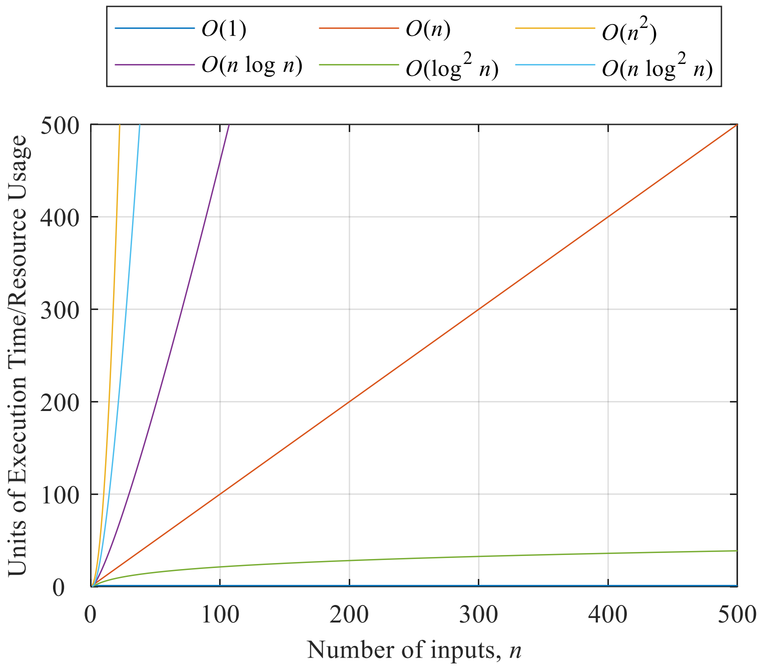


Figure 4.1: Common growth functions for sorting algorithm execution time and resource usage

Since Figure 4.1 is for comparison purposes only, the *y*-axis scale is arbitrary and does not correspond to a physical quantity for execution time or resource usage. To obtain an absolute value for either quantity at a specific value of n, the chosen sorting algorithm must be implemented on a control hardware target and its execution time and resource usage measured. Scaling coefficients for the growth function can then be calculated and used to estimate the execution time or resource usage for different values of n.

4.1.2 Time Complexity

Time complexity is dependent upon the number of basic operations (compare, swap, copy) performed by the sorting algorithm, which are dictated by the algorithm structure. Time complexity growth rates can be defined as best-, average-, and worst-case. For many common sorting algorithms, the best-case execution time will typically occur when the input set is already fully sorted, whilst the worst-case will occur when the input set is reversed with respect to the direction of the $\mathrm{sort}()$ operation [65].

In the case of sorting algorithms for CBC where the execution window is time-limited, the worst-case execution time on a particular control hardware target is of primary interest, since exceeding the available execution window duration may lead to degraded capacitor balancing performance. The best- and worst-case time complexities for several sorting algorithms commonly used in the literature on sorting-based CBC methods [49, 50, 53, 66-72] are shown in Table 4.1. These algorithms have been selected for further analysis.

Sorting Algorithm	**Best-Case Time Complexity**	**Worst-Case Time Complexity**
Bubble Sort	$O(n)$	$O(n^2)$
Insertion Sort	$O(n)$	$O(n^2)$
Merge Sort	$O(n \log n)$	$O(n \log n)$
Quick Sort	$O(n \log n)$	$O(n^2)$
Bitonic Merge Sort	$O(\log^2 n)$	$O(\log^2 n)$
Odd-Even Merge Sort	$O(\log^2 n)$	$O(\log^2 n)$

Table 4.1: Best- and worst-case time complexities for several common sorting algorithms for sorting-based CBC methods [73]

Three algorithms (bubble sort, insertion sort, and quick sort) have quadratic ($O(n^2)$) worst-case time complexities. This may appear to discount them when compared to the other algorithms presented, which have smaller growth rates. It is important to note however, that two other key factors will affect the execution time of each algorithm:

- Implementation specifics in code, such as language-specific optimisations or recursion.
- Adaptability of the sorting algorithm and the existing order of the input set.

Adaptability refers to the ability of a sorting algorithm to take advantage of existing order in the input set. Both bubble sort and insertion sort are classed as adaptive sorting algorithms and will have a shorter execution time when the input set is more ordered. Merge sort and quick sort are not adaptive however, so cannot take advantage of the existing order of the input set [65]. To obtain a reliable estimate of execution time, each sorting algorithm must be implemented in code running on a control hardware target, then tested with a representative pattern of SM capacitor voltages.

4.1.3 Space Complexity

As stated previously, space complexity describes how the resource usage of a sorting algorithm scales as n is increased. In the case of sorting algorithms implemented on microcontroller- or microprocessor-based control hardware targets, the resource consumed by the sorting algorithm is memory. For sorting algorithms implemented on a field-programmable gate array (FPGA), the resources used by the sorting algorithm are logic elements on the device, such as flip-flops (FF) and lookup tables (LUT), which are used as memory and to implement processing operations.

Table 4.2 lists the space complexity of the six sorting algorithms presented previously in Table 4.1 and can be compared graphically using the curves in Figure 4.1. Unlike time complexity, space complexity is specified as a worst-case value only and is not affected by the characteristics of the input set such as existing ordering.

Sorting Algorithm	**Worst-Case Space Complexity**
Bubble Sort	$O(1)$
Insertion Sort	$O(1)$
Merge Sort	$O(n)$
Quick Sort	$O(\log n)$
Bitonic Merge Sort	$O(n \log^2 n)$
Odd-Even Merge Sort	$O(n \log^2 n)$

Table 4.2: Worst-case space complexities for several common sorting algorithms for sorting-based CBC methods [73]

It is immediately obvious by comparing Table 4.1 and Table 4.2 that sorting algorithms with a low (better) time complexity typically have a high (worse) space complexity and vice-versa. This leads to a trade-off between execution time and resource usage when choosing a suitable sorting algorithm for the CBC loop.

4.1.4 Sequential vs. Parallel Sorting Algorithms

Sorting algorithms can also be classified based upon whether the algorithm performs operations on input set values in sequence or in parallel. In the case of comparison-based sorting algorithms such as those listed in Table 4.1, the main operation performed by the algorithm is the compare-swap operation. This operation is defined as shown in Figure 4.2, for two input values, x and y.

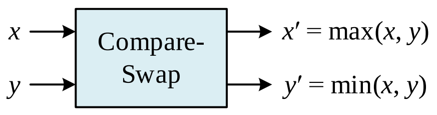


Figure 4.2: Compare-swap operator

Two categories can be defined at either end of a sliding scale based upon whether the algorithm performs compare-swap operations sequentially or in parallel:

- Fully sequential: no parallelisable compare-swap operations, only a single operation can be performed at a time. Pairs of input values must be operated on in sequence.
- Fully parallel: two or more compare-swap operations can be performed in parallel. Two or more input value pairs can be operated on simultaneously, independent of other values.

Classifying sorting algorithms as sequential or parallel corresponds closely with the mapping of a particular sorting algorithm onto best-suited processing hardware, which can also be categorised as operating sequentially or in parallel:

- Sequential: central processing unit (CPU) or microcontroller-based hardware. Each processing core can only operate upon a single pair of values per clock cycle. Limited parallelisation is achieved on a single core device by sharing the processor core across multiple tasks (multiplexing), or on a multi-core device by allocating operations to different processor cores.
- Parallel: FPGA and digital signal processor (DSP)-based hardware can operate upon multiple pairs of values in parallel per clock cycle due to the potential for massive duplication of functional blocks (eg. compare-swap operators) in fixed logic hardware.

Classification of sorting algorithms according to sequential or parallel operation is less well-defined than time and space complexity comparison metrics since it is possible for sequential sorting algorithms to be implemented on parallel hardware and vice-versa. Furthermore, some traditionally sequential sorting algorithms have been optimised for implementation on parallel processing hardware. Analysis of the performance of optimised variants of the sorting algorithms listed in Table 4.1 is outside the scope of this research and only the traditional implementation of each algorithm will be considered. Pseudocode for the variants of the sorting algorithms as implemented in this work is provided in Appendix A for reference purposes.

Figure 4.3 classifies the sorting algorithms listed in Table 4.1 and the processing hardware types on the sliding scale described previously. Bubble sort and insertion sort are difficult to parallelise since these algorithms fundamentally operate on the whole input set without dividing it into smaller sub-sets. Merge sort and quick sort support limited parallelisation due to their divide-and-conquer approach to the sorting problem. Both algorithms divide the input set into partitions of decreasing size before sorting each partition; this process can be carried out recursively in parallel.

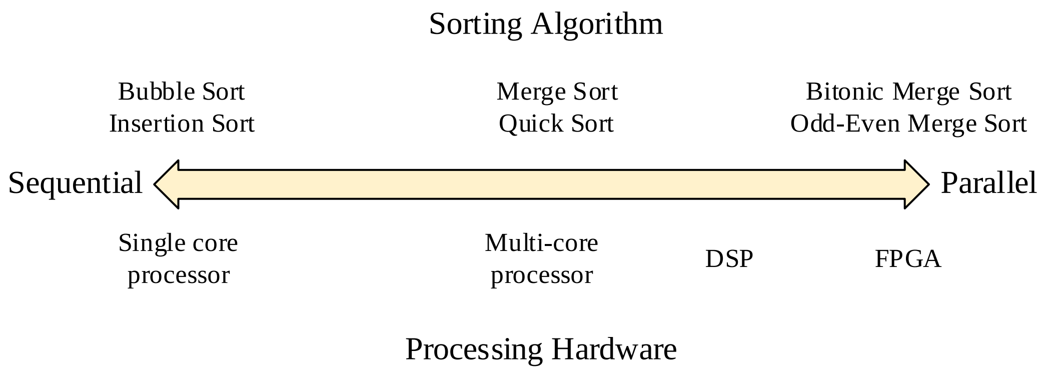


Figure 4.3: Categorisation of sorting algorithms and processing hardware on sequential vs. parallel sliding scale

4.2 Sorting Networks

Bitonic merge sort and odd-even merge sort are both types of sorting network and are inherently parallel in operation. Unlike classical comparison-based sorting algorithms which perform a variable number of compare-swap operations, sorting networks have a fixed sequence of compare-swap operations which operate on several pairs of values independently, in parallel [74]. As a result, they are well-suited to implementation on parallel processing hardware such as FPGAs and typically exhibit a lower execution time when compared to classical sorting algorithms.

4.3 Summary

This chapter has provided an introduction to sorting algorithms which serves as a foundation for the rest of this thesis. The comparison metrics described in this chapter will be used in later chapters when comparing sorting algorithm performance across different hardware and software implementations. The sorting algorithms (bubble sort, insertion sort, merge sort, quick sort) and sorting networks (bitonic merge sort, odd-even merge sort) reviewed in this chapter were chosen based upon a review of sorting algorithms for CBC and are the most commonly referenced algorithms in the existing literature.

Sorting algorithms have been and continue to be the subject of intensive research in the field of computer science. Further research avenues into sorting algorithms for CBC may however focus upon novel algorithms such as those proposed in the literature published in the field of computer science.

5 Simulation Model Overview

This chapter provides an overview of the PSCAD/EMTDC simulation model which was used to generate synthetic SM capacitor voltage data for the investigation into sorting algorithm execution time presented in Chapter 6. To ensure that the results from the investigation are representative of industrial-scale HVDC MMCs which typically have several hundred SMs per arm, an MMC simulation model with greater than 200 levels was required.

To minimise the time spent developing and verifying a new simulation model, an open-access MMC model developed by The National HVDC Centre and the University of Strathclyde was used [75]. The model is one of several developed as part of a project titled: “Developing Open-Source Converter Models” [22] and implements a 351-level MMC using the half-bridge SM configuration. A range of models with different circuit topologies and number of levels are available; the 351-level HB-SM model was chosen since has an industrially representative number of SMs per phase and the HB-SM topology has a simpler control structure compared to full-bridge or hybrid SM topologies. A more detailed description of the simulation model can be found in [76-80].

The circuit structure and converter ratings of the MMC model are introduced first in this chapter and the fault locations used in the simulation scenarios in Chapter 6 are shown. A brief overview of the control structure implemented in the model is then provided, followed by a description of the custom capacitor balancing control component which has been developed to enable the research carried out in Chapter 6.

5.1 Circuit Structure

The circuit structure of the MMC model is shown in Figure 5.1 and the system parameters are listed in Table 5.1. As stated previously, the MMC is configured as a 351-level three phase HB-MMC with 350 SMs per arm. On the AC side, the converter is connected to the AC ‘grid’ via an interfacing transformer with a voltage ratio of 360/400 kV. Since the dynamics of the AC system are not the focus of this work, the AC grid is modelled as an ideal AC source behind a fixed impedance which is configured to provide a short-circuit ratio (SCR) of 10. The voltage, current, and power measurement locations and polarities are also shown in Figure 5.1; these correspond to the measurements shown in the figures in Chapter 6.

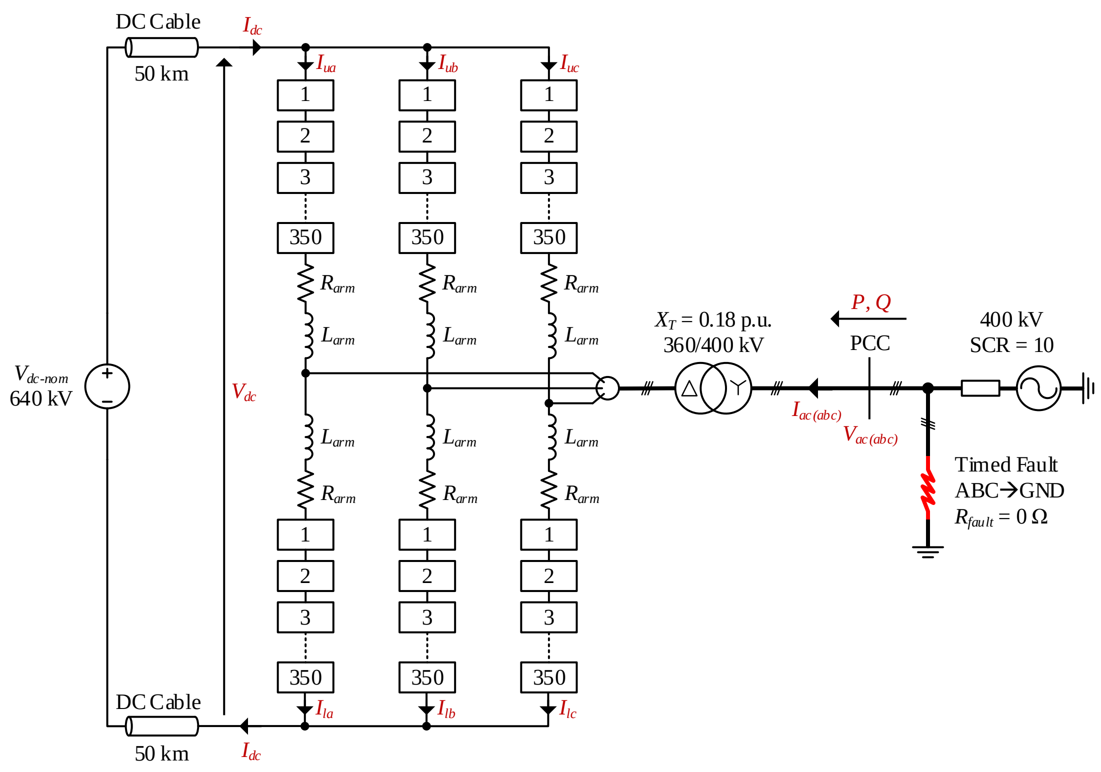


Figure 5.1: MMC model circuit diagram

Parameter	Value	Units
MMC rated apparent power, S	1265	MVA
MMC rated active power, P	1200	MW
MMC rated reactive power, Q	400	MVAr
MMC rated AC output voltage (line-line)	360	kV
Arm inductance, L_{arm}	42	mH
Arm resistance, R_{arm}	0.08	Ω
Number of SMs per arm, N_{SM}	350	
SM capacitance, C_{SM}	11	mF
Nominal DC bus voltage, V_{dc}	640 (± 320)	kV
AC system short-circuit ratio (SCR)	10	
AC system nominal frequency, f_0	50	Hz
Transformer rated apparent power, S_T	1265	MVA
Transformer voltage ratio	400/360	kV
Transformer leakage reactance, X_T	0.18	p.u.
Transformer resistance	0.004452	p.u.

Table 5.1: MMC model parameters, reproduced from [80]

On the DC side, the converter is connected to the DC terminal via a 50 km underground cable modelled using a frequency-dependent phase method. The DC source is modelled as an ideal voltage source with a 0 Ω internal resistance. Modelling of the DC terminal and underground cable dynamics is not the focus of this work and will not be discussed further.

To allow testing for a fault scenario in Chapter 6, a three phase-to-ground symmetrical fault at the point of common coupling (PCC) was added to the simulation model. The fault resistance, R_{fault}, was set to 0 Ω across all three phases and the converter control was configured not to block the SMs during the fault. When not simulating a fault scenario the timed fault block is disabled.

5.2 Control Structure

A high-level block diagram of the control structure of the MMC model is provided in Figure 5.2, showing the controllers which are implemented in the model. A more detailed description of the operation of each controller can be found in [80]. The converter operates in active/reactive power (PQ) control mode using a PQ set-point which can be configured using interactive user controls within the simulation, or a pre-configured set-point profile.

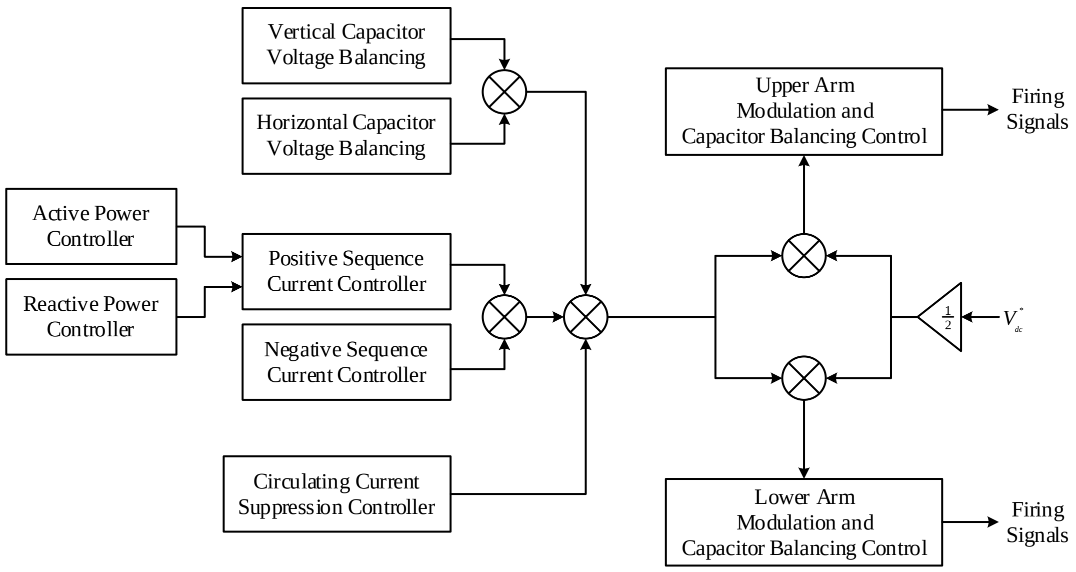

Vertical Capacitor Voltage Balancing
Horizontal Capacitor Voltage Balancing
Active Power Controller
Reactive Power Controller
Positive Sequence Current Controller
Negative Sequence Current Controller
Circulating Current Suppression Controller
Upper Arm Modulation and Capacitor Balancing Control
Lower Arm Modulation and Capacitor Balancing Control
Firing Signals
Firing Signals
$\frac{1}{2}$
V_{dc}^{*}

Figure 5.2: Block diagram showing MMC model control structure, adapted from [80]

In the control structure shown in Figure 5.2, the active/reactive power controllers and positive/negative sequence current controllers are implemented using indirect control in the dq domain. The active power controller acts upon the d-axis current reference passed to the positive sequence current controller, whilst the reactive power controller modifies the q-axis current reference. The output of the positive sequence current controller is a dq voltage reference which is then converted back into the abc domain using the inverse Clarke-Park transformation.

A negative sequence current controller is also implemented in the model to ensure that the converter response remains controlled during unbalanced operation, such as during asymmetric AC network faults. In this work, it is assumed that maintaining balanced three phase AC currents is the control objective, therefore the d- and q-axis negative sequence current references are both set to zero. The output of the negative sequence current controller is also a dq voltage reference which is converted back into the abc domain and summed per-phase with the reference from the positive sequence controller.

The vertical capacitor voltage balancing controller is implemented to balance the stored energy between the upper and lower arms in a phase. Similarly, the horizontal capacitor voltage balancing controller balances the stored energy across the three phases in the converter. These controllers facilitate the regulation of stored energy across the six converter arms and improve the dynamic response of the MMC to active power set point changes [80].

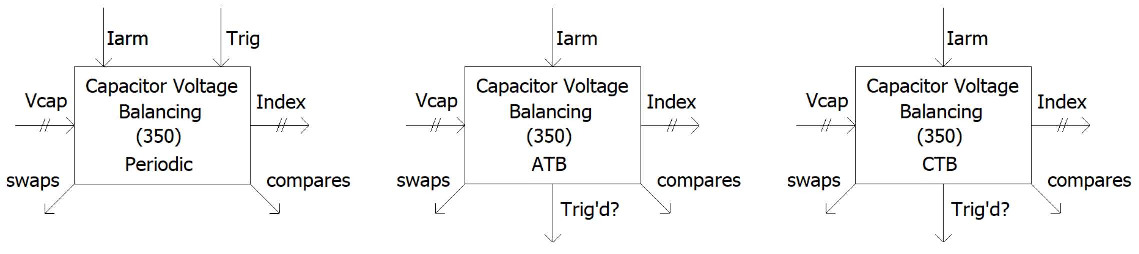
The circulating current suppression controller (CCSC) operates to suppress the AC component of the circulating current, which can increase converter losses and SM capacitor voltage ripple [30]. In the model, the CCSC is implemented in the abc domain as a proportional-resonant (PR) controller [80].

The output voltage references from the two current controllers, horizontal and vertical capacitor voltage balancing controllers, and the CCSC are summed with the DC bus voltage set point, V_{dc}^{*}, to generate an upper and lower arm voltage reference for each phase. These references are passed to the modulation and capacitor balancing control (CBC) loop. Nearest level control (NLC) output modulation is implemented in this model, since it is simpler to implement for MMCs with a large number of SMs when compared to pulse-width modulation (PWM)-based methods. The operation of the CBC component is described in the following section.

5.3 Capacitor Balancing Control Custom Component

To facilitate research into the effect of different CBC methods upon sorting algorithm execution time as detailed in Chapter 6, a custom PSCAD/EMTDC capacitor balancing control component was developed for this work. The standard CBC component provided in the model was unsuitable for this work since it only implements a single CBC method (average tolerance band) and is not completely documented in the reports describing the model.

For the investigation into sorting algorithm execution time, periodic, average tolerance band (ATB), and cell tolerance band (CTB) CBC methods were chosen for testing and were implemented in the custom component. Periodic CBC is triggered at a fixed sampling frequency supplied by a pulse generator block in PSCAD/EMTDC. In the ATB method, balancing is triggered when the capacitor voltage of one or more SMs exceeds a tolerance band around the instantaneous mean SM capacitor voltage ($\overline{V_{cap}}[k]$). In the CTB method, balancing is triggered when the capacitor voltage of one or more SMs exceeds a fixed upper or lower limit. The CBC method in use can be changed prior to starting the simulation by changing the component parameters – it is not necessary to replace the component when changing CBC methods. The graphic overlay of the custom PSCAD/EMTDC developed in this PhD is shown in Figure 5.3(a) to (c) for the three CBC methods tested.


Iarm
Trig
Vcap
Capacitor Voltage Balancing (350) Periodic
Index
swaps
compares
Iarm
Vcap
Capacitor Voltage Balancing (350) ATB
Index
swaps
compares
Trig'd?
Iarm
Vcap
Capacitor Voltage Balancing (350) CTB
Index
swaps
compares
Trig'd?

(a) (b) (c)

Figure 5.3: PSCAD/EMTDC custom capacitor balancing control component mask configured for (a) periodic, (b) ATB, and (c) CTB CBC methods

The *Vcap* and *Iarm* ports are inputs for the list of SM capacitor voltages and arm current, respectively. The *Index* port is an output and provides the list of sorted SM identifiers to the firing block component, which generates firing signals for all SMs in an arm using the list of SM identifiers and SM insertion index, N_{on}. Two additional output ports labelled *swaps* and *compares* can be enabled if required; these output the number of swap and comparison operations performed by the CBC sorting algorithm when the CBC block is triggered. Sorting algorithm execution time is typically proportional to the number of compare and swap operations carried out, however these output ports were not used since execution time was measured directly for the results presented in Section 6.3 and Section 6.4.

The *Trig'd?* output port shown in Figure 5.3(b) and (c) can be enabled to provide an output signal which is set to '1' in the simulation timesteps where the CBC loop is triggered and '0' otherwise. This port is only available when ATB or CTB is chosen as the CBC method, since these methods do not have a fixed sampling frequency; the sampling clock is generated internal to the CBC component instead. For the analysis carried out in Chapter 6, the ability to log this internal sampling clock was required. The periodic CBC variant of the component has a trigger input port labelled *Trig* which is used to provide a periodic sampling clock to the CBC component.

The tolerance bands used for ATB and CTB CBC methods can be configured prior to starting the simulation by modifying the component parameters. Where the ATB method is used, the tolerance band, δ_{ATB}, around the instantaneous mean SM capacitor voltage, $\overline{V_{cap}}[k]$, can be specified in kV or as a percentage of $\overline{V_{cap}}[k]$. For the CTB method, the upper and lower limits on SM capacitor voltage are specified in kV.

Bubble sort is used as the sorting algorithm in the custom component since it has a low implementation effort and its contribution to the overall simulation runtime is small relative to the other calculations performed by the EMTDC solver. Bubble sort works by repeatedly iterating over the input array, comparing adjacent elements, and swapping them if they are in the wrong order [65]. It is important to note that the execution time results presented in Chapter 6 were generated by sorting the list of SM capacitor voltages offline in real-world control hardware – that is, the sorting algorithm used in the CBC custom component is only used during simulation to ensure that capacitor voltages remain balanced.

5.3.1 Capacitor Balancing Control Method Implementation

The sequence of operations implemented in each of the CBC methods provided by the custom component is shown in flowchart form in Figure 5.4 to Figure 5.6 for reference purposes and is described in the following sub-sections. FORTRAN code for each CBC method is included in Appendix B, and the custom PSCAD/EMTDC component can be downloaded from the repository in [26].

Periodic

The sequence of operations for the periodic CBC method is shown in the flowchart in Figure 5.4. During each simulation timestep when the CBC component code is executed by the simulator, the ordered list of SM identifiers from the previous timestep is loaded first. For periodic CBC, the *Trig* input terminal is then checked to determine if the CBC loop should be triggered. When *Trig* = F (false), capacitor balancing is not triggered and the list of SM identifiers from the previous timestep is used to update the list of SM identifiers for the current timestep. When *Trig* = T (true), capacitor balancing is triggered, and the SM identifiers are sorted according to the measured SM capacitor voltages and arm current direction. Where the arm current is greater than zero, the capacitors of inserted SMs will be charged and as a result, SMs with lower voltages should be prioritised for insertion to ensure capacitor voltages remain balanced. This is achieved by sorting the SM capacitor voltages in ascending order and placing the corresponding SM identifiers into the array first. Conversely, when the arm current is less than zero, inserted SMs will discharge and SMs with higher voltages should be prioritised for insertion. This is achieved by sorting the list of SM capacitor voltages in descending order.

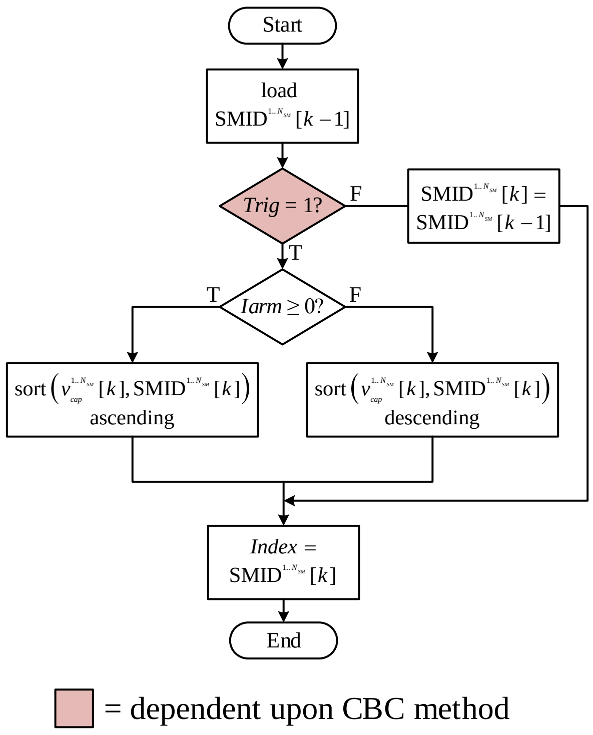

Start
load $\mathrm{SMID}^{1..N_{SM}}[k-1]$
$Trig = 1?$
F
$\mathrm{SMID}^{1..N_{SM}}[k] = \mathrm{SMID}^{1..N_{SM}}[k-1]$
T
T
$Iarm \geq 0?$
F
$\mathrm{sort}\left(v_{cap}^{1..N_{SM}}[k], \mathrm{SMID}^{1..N_{SM}}[k]\right)$ ascending
$\mathrm{sort}\left(v_{cap}^{1..N_{SM}}[k], \mathrm{SMID}^{1..N_{SM}}[k]\right)$ descending
$Index = \mathrm{SMID}^{1..N_{SM}}[k]$
End
🟥 = dependent upon CBC method

Figure 5.4: Flowchart showing periodic capacitor balancing control method sequence of operations

The resulting list of SM identifiers generated during the current timestep is then assigned to the *Index* output terminal which is connected to the firing block component.

Average Tolerance Band

The primary difference between the CBC methods implemented in this work is the conditions under which the CBC loop is triggered. This is shown by the decision block in Figure 5.4 highlighted in red. The remaining sequence of operations in the CBC loop are identical across the three CBC methods, therefore these will not be repeated. The trigger decision tree for the ATB method is shown in Figure 5.5.

Upon entering the ATB method decision tree, the *Trig'd* output is initialised to '0' for the eventuality that the CBC loop is not triggered. The instantaneous mean capacitor voltage, $\overline{V_{cap}}[k]$, is then calculated using the measured SM capacitor voltages for the current timestep. The calculation of $\overline{V_{cap}}[k]$ was described in more detail in Section 2.1.1.

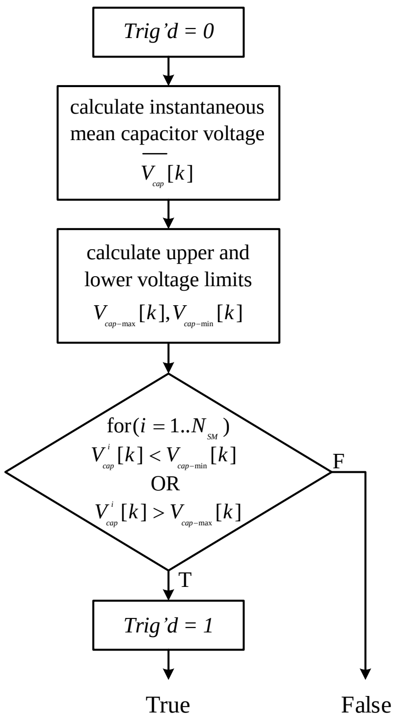

Trig'd = 0
calculate instantaneous mean capacitor voltage $\overline{V_{cap}}[k]$
calculate upper and lower voltage limits $V_{cap-\max}[k], V_{cap-\min}[k]$
for($i = 1..N_{SM}$) $V^i_{cap}[k] < V_{cap-\min}[k]$ OR $V^i_{cap}[k] > V_{cap-\max}[k]$
F
T
Trig'd = 1
True
False

Figure 5.5: Flowchart showing trigger decision tree for average tolerance band CBC method

The upper and lower capacitor voltage limits for the current timestep, $V_{cap-max}[k]$, and $V_{cap-min}[k]$, are then calculated using $\overline{V_{cap}}[k]$ and the tolerance band (δ_{ATB}) specified in the component parameters dialog box according to Equation 5.1 and Equation 5.2.

$$V_{cap-max}[k] = \overline{V_{cap}}[k] + \delta_{ATB} \tag{5.1}$$

$$V_{cap-max}[k] = \overline{V_{cap}}[k] - \delta_{ATB} \tag{5.2}$$

The measured SM capacitor voltages for the current timestep are then compared with the upper and lower capacitor voltage limits in a `for()` loop. If one or more SM capacitor voltages exceed the upper or lower band limits, the CBC loop is triggered, the *Trig'd* output port is set to '1', and capacitor balancing is carried out as described previously. In the situation where all SM capacitor voltages are inside the tolerance band, the CBC loop is not triggered.

Cell Tolerance Band

The decision tree for the CTB method is shown in Figure 5.6. Unlike the ATB method, the tolerance band is fixed and is not dependent upon the mean SM capacitor voltage. At each timestep, the *Trig'd* output is initialised to '0' for the scenario where CBC loop is not triggered. The measured SM capacitor voltages for the current timestep are then compared with the upper and lower capacitor voltage limits and CBC is triggered if an individual SM capacitor voltage exceeds the upper or lower band limits. As with the ATB method, the *Trig'd* output port is set to '1' when the CBC loop is triggered.

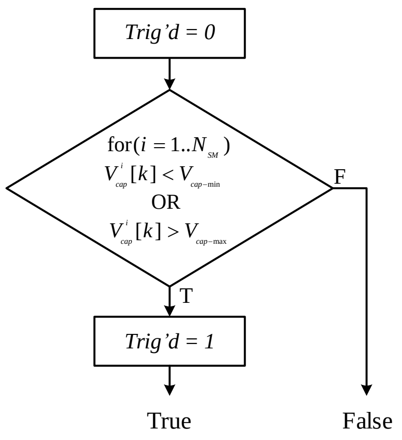


Figure 5.6: Flowchart showing trigger decision tree for cell tolerance band CBC method

5.4 Summary

This chapter has provided an overview of the simulation model used to generate synthetic SM capacitor voltages for the investigation into sorting algorithm execution time. A custom CBC component has been developed in PSCAD/EMTDC to implement the CBC methods of interest, which have not yet been implemented in a publicly-available component [26]. The three CBC methods implemented in the custom component have been validated against the SM capacitor voltage curves presented in [31] for periodic CBC and [43] for ATB and CTB methods and found to be in agreement. The sequence of operations for each CBC method have been described as a basis for the results presented in later chapters.

6 Sorting Algorithms for Capacitor Balancing Control

One of the objectives of this research is to assess the effects of software implementation of control loops upon execution delay and resource usage on real-world control hardware. Of particular interest in this work are the sorting algorithms used by the MMC internal capacitor balancing control (CBC) loop.

In this chapter, the motivations for investigating the CBC loop are outlined first, followed by a description of the programming languages and control hardware target platforms upon which the selected sorting algorithms have been implemented. Sorting algorithm execution time measurements are then presented and discussed for three different CBC methods and simulation scenarios. The selected sorting algorithms were then programmed using three implementation methods on FPGA-based control hardware and the logic resource usage measured. These results are also presented and discussed in this chapter. Several conclusions and recommendations are then developed which can be used to guide the selection and implementation process for sorting algorithms for CBC.

6.1 Motivations

The motivations for investigating the CBC loop can be split into two categories: firstly, those related to the execution time of the CBC loop and the underlying sorting algorithm, and secondly, those related to the resource usage of a particular sorting algorithm when implemented in software running on real-world control hardware. Both categories are closely linked, and a balance must be struck between the upper limit on sorting algorithm execution time and resource usage when implemented on a control hardware target.

Comparison of sorting algorithm performance using the existing literature is difficult due to the range of programming languages and control hardware targets used. The results from this work will address this and provide a direct comparison between several sorting algorithms implemented on two control hardware targets in three programming languages.

6.1.1 Sorting Algorithm Execution Time

The primary motivation for measuring sorting algorithm execution time is the non-deterministic (variable) execution time of the classical sorting algorithms which are commonly used to sort SM capacitor voltages in the CBC loop. Typically, the CBC loop must run in hard real-time, with a time-limited execution window (control cycle) within which all processing operations must be completed, including sorting. The execution window is determined by the chosen CBC method, with some methods requiring more frequent sorting of SM capacitor voltages in a shorter execution window.

Non-deterministic execution of the sorting algorithm may lead to the execution time exceeding the CBC loop control cycle period. This will lead to a delay in updating the firing signals to the SMs for balancing purposes. Alternatively, it may be necessary to increase the CBC loop control cycle period to ensure that the sorting algorithm completes before the SM firing signals are updated. Either of these factors may lead to degraded capacitor balancing performance and an increase in SM capacitor ripple voltage, reducing the lifespan of the SM capacitor [81].

6.1.2 Sorting Algorithm Implementation Resource Usage

The CBC loop and corresponding sorting algorithm are implemented alongside other converter control functions on the same control hardware target [82]. At the arm control level, one or more FPGAs are typically used as the control hardware platform due to the ability to perform many complex processing operations in parallel. FPGAs have a finite number of logic resources in the form of flip-flops (FF) and lookup tables (LUT) which can be used to implement processing operations.

To ensure that there are sufficient logic resources available on the FPGA to implement CBC using a chosen sorting algorithm alongside other control functions, an estimate of the logic resource usage of the sorting algorithm is required. Sorting algorithm resource usage is typically dependent upon two factors: the number of inputs to the sorting algorithm, n, and the structure of the chosen sorting algorithm. In the case of sorting algorithms for CBC, n is equal to the number of submodules per arm, N_{SM}. Furthermore, whilst sorting algorithms with a deterministic execution time do exist, these algorithms also typically consume more FPGA logic resource due to their structure. As a result, it was decided to measure the logic resource usage of the sorting algorithms of interest for a range of values of n, when implemented on an FPGA using three different programming languages.

6.2 Sorting Algorithm Implementation

As stated in Section 6.1, comparing sorting algorithm execution time and resource usage using the existing literature is difficult due to the wide range of control hardware targets and programming languages used to implement the algorithms. This section provides an overview of the control hardware targets and programming languages used to implement the sorting algorithms chosen for analysis in this work.

6.2.1 Control Hardware Targets

To facilitate a direct comparison of sorting algorithm execution time on CPU- and FPGA-based control hardware targets, the sorting algorithms listed in Table 4.1 were implemented on two control hardware targets: a National Instruments (NI) PXIe-8102 embedded controller [83] (Intel x86 CPU-based) and a NI PXIe-7857R FPGA [84] (Xilinx Kintex-7 FPGA). These platforms were chosen since they were already available in the converter hardware prototype (CHP) PXIe control chassis and are representative of the processing platforms used widely in industrial control hardware. The key specifications of both platforms are listed in Table 6.1.

Platform	**Architecture Type**	**Key Specifications**
NI PXIe-8102	x86 CPU (sequential)	• 1.9 GHz dual-core Intel Celeron T3100 CPU [85] • 2 GB RAM • Operating system: Phar Lap ETS RTOS
NI PXIe-7857R	FPGA (parallel)	• Xilinx Kintex-7 160T FPGA [86] • 25,350 logic slices • 202,800 flip-flops • 101,400 lookup tables • f_{clock} = 100 MHz (used for testing)

Table 6.1: Summary of key specifications for control hardware targets

The Intel Celeron T3100 CPU used in the NI PXIe-8102 was launched in 2008 [85] and has since been discontinued. Whilst the specifications of this platform lag those of more modern CPUs, it can still be used to provide a reliable estimate of sorting algorithm execution time on CPU-based control hardware targets. This is due to the simple structure of the algorithms under test and the fact that the algorithms have not been optimised for a particular CPU.

The Xilinx Kintex-7 160T FPGA used in the NI PXIe-7857R is still in production as part of Xilinx's mid-range FPGA portfolio. The -160T variant of the device is positioned in the bottom quarter of the Kintex-7 series devices and contains 25,350 logic 'slices' each of which contains eight flip-flops and four lookup tables [86]. Industrial control hardware targets may use FPGAs with more slices or devices from a different product line such as the Virtex-7 series. The underlying structure of the logic slices is the same across both Kintex-7 and Virtex-7 series, therefore the execution time and logic resource usage results are applicable to control hardware targets which use other Xilinx FPGAs. In this work, a FPGA clock frequency, f_{clock}, of 100 MHz was used to calculate the sorting algorithm execution time in seconds from the number of clock cycles taken to execute the algorithm.

6.2.2 Programming Methods

Both the PXIe-8102 and PXIe-7857R control hardware targets are programmed primarily in LabVIEW, a high-level graphical programming language which has similarities to MATLAB/Simulink. A more detailed overview of the LabVIEW programming language and the different variants for each hardware target is provided in Section 8.1. The target hardware platforms can also be programmed using low-level languages such as the C programming language for the PXIe-8102 and Verilog/VHDL for the PXIe-7857R. The programming methods used in this work are outlined and compared in the following sub-sections.

LabVIEW Standard

LabVIEW 'Standard' as termed in this work is used to describe sorting algorithm implementations which are platform-agnostic and will run on either CPU or FPGA-based platforms without requiring modification of the LabVIEW block diagram code. The characteristics of this programming method can be summarised as follows:

- Uses LabVIEW block diagram constructs which are available across all control hardware targets, such as standard loops (`for()`, `while()` loops etc.).
- Highest level of programming abstraction possible in LabVIEW.
- Relies heavily upon LabVIEW run-time environment or FPGA compiler to translate a high-level block diagram into target-specific executable code.
- Requires minimal specific understanding of the underlying architecture of the hardware platform to produce functional code, therefore is relatively user-friendly.

Use of a high-level abstraction will typically reduce sorting algorithm performance and increase the logic resource usage when implemented on an FPGA due to the additional translation steps required to produce executable code from the high-level block diagram.

LabVIEW FPGA Optimised

LabVIEW FPGA ‘Optimised’ as termed in this work describes sorting algorithm implementations which are specific to the PXIe-7857R FPGA control hardware target. Sorting algorithms implemented using this method are still programmed using a graphical block diagram as with other LabVIEW variants, however, can be distinguished from LabVIEW Standard implementations by the following characteristics:

- Uses FPGA-specific LabVIEW block diagram constructs (eg. single-cycle timed loops) and hand-coded finite state machines to control algorithm execution.
- Code cannot be executed on CPU-based platforms without modification.
- Requires an understanding of the underlying architecture of the FPGA and how FPGAs execute processing operations to produce functional code.

LabVIEW FPGA-specific block diagram constructs map more directly onto the underlying architecture of the FPGA and provide the programmer with more control over how the algorithm is implemented. When compared to the LabVIEW Standard implementation on an FPGA, the FPGA Optimised route relies less upon the LabVIEW FPGA compiler to translate the block diagram into executable code (or logic) on the FPGA. This will typically reduce FPGA logic resource usage and execution time of the sorting algorithm. This is due to bypassing the automated synchronisation and flow control logic inserted by the LabVIEW FPGA compiler when using LabVIEW Standard on an FPGA [87]. Producing functional code does however require a deeper understanding of FPGA programming techniques, therefore this programming method is less user-friendly.

Verilog

Verilog is a text-based hardware description language (HDL) which is used to describe processing operations to be implemented as logic circuits on an FPGA. It is part of a family of HDLs including VHDL and SystemVerilog which are used in conjunction with a synthesis tool to translate the design onto elementary logic primitives on the FPGA such as logic gates and flip-flops.

Whilst Verilog and other HDLs can be used to directly synthesise logic primitives at the lowest level of abstraction, more commonly, register-transfer level (RTL) abstraction is used. In RTL design, a digital logic circuit is described as a set of registers and a set of logic functions which describe the operations performed on the data in the registers. The registers are implemented as synchronous logic (flip-flops) on the FPGA whilst the logic functions are implemented as combinatorial logic [88]. This abstraction has been used to implement the sorting algorithms in this work.

Verilog is the lowest level programming method used to implement the sorting algorithms chosen for analysis in this work and is FPGA-specific. When compared to both LabVIEW programming methods, it can produce highly optimised logic on the FPGA due to removing the additional layers of abstraction introduced by LabVIEW. It does however require an in-depth understanding of FPGA programming techniques and is the least user-friendly of the three programming methods.

The key comparators described in the previous three sub-sections are summarised in Table 6.2. The terms used in the table are relative to the other programming methods used.

Programming Method	**Abstraction Level**	**Performance**	**Programming Difficulty**
LabVIEW Standard	High	Worst	Low
LabVIEW FPGA Optimised	Medium	Better	Medium
Verilog	Low	Best	High

Table 6.2: Comparison of sorting algorithm programming methods

6.2.3 Sorting Algorithm Implementation Summary

An overview of the combinations of sorting algorithms and programming methods which have been tested is provided in Table 6.3. All algorithms were tested for n = 4, 8, 16, 32, 128, and 256 except for bitonic merge sort and odd-even merge sort, which were only tested to n = 64 and n = 32 respectively. A maximum value of n = 256 was chosen as a trade-off between the range of the number of submodules per arm, N_{SM}, in a typical HVDC-scale MMC and the implementation effort required to test for higher values of n. Increasing n in powers of two was chosen so that a larger range of values could be tested whilst reducing the implementation effort required to test linearly spaced values.

Bitonic merge sort and odd-even merge sort were only tested to n = 64 and n = 32 respectively due to the significant increase in effort required to manually implement the sorting network structure for larger values of n. This in itself is a disadvantage of sorting networks. Whilst software tools have been created to automatically generate sorting networks with arbitrary numbers of inputs [89], these tools were not used since they generate Verilog source code for a specific FPGA platform and do not support LabVIEW-based programming methods.

Merge sort and quick sort were not implemented on the FPGA platform since both algorithms use recursion, which is not natively supported by FPGA hardware targets. The input to each sorting algorithm is an array of 10-bit SM capacitor voltages (corresponding to a SM capacitor voltage analogue-to-digital converter resolution of 10 bits) and an 8-bit SM identifier (maximum N_{SM} = 256). The trends in the results presented in the following sections are not tightly coupled to the use of 10-bit SM capacitor voltage readings and 8-bit SM identifiers, however the resource usage will change where different numbers of bits are used to represent SM capacitor voltage readings or SM identifiers.

Programming Method	**Sorting Algorithm**											
	Bubble Sort		**Insertion Sort**		**Merge Sort**		**Quick Sort**		**Bitonic Merge Sort**∗		**Odd-Even Merge Sort**∗	
	CPU	FPGA	CPU	FPGA	CPU	FPGA	CPU	FPGA	CPU	FPGA	CPU	FPGA
LabVIEW Standard	✓	✓	✓	✓	✓	✗†	✓	✗†	✓	✓	✓	✓
LabVIEW FPGA Optimised	▨	✓	▨	✓	▨	✗†	▨	✗†	▨	✓	▨	✓
Verilog	▨	✓	▨	✓	▨	✗†	▨	✗†	▨	✓	▨	✓

Table 6.3: Matrix of sorting algorithms, control hardware targets and programming languages tested

Notes:

▨ Grey shaded cells indicate that this combination of control hardware target and programming language is not possible.

† Merge sort and quick sort are unsuitable for implementation on FPGA-based architectures since the standard implementation of both algorithms requires recursion, which is not supported by FPGAs.

∗ Bitonic merge sort was only implemented to $n = 64$ whilst odd-even merge sort was only implemented to $n = 32$. This is due to the significant increase in effort to manually implement the sorting network structure at higher values of n.

6.3 Sorting Algorithm Execution Time Range

As stated previously, for sorting-based CBC methods, the sorting algorithm is a major source of execution delay relative to the total delay of the CBC loop and is often non-deterministic. If the execution time of the sorting algorithm exceeds the control cycle period of the CBC loop, this may cause degraded capacitor balancing performance. Of particular interest is the maximum possible execution time of the sorting algorithm since this will dictate the minimum control cycle period achievable with a particular combination of sorting algorithm and control hardware.

To determine the execution time range of the sorting algorithm implementations listed in Table 6.3, each algorithm was tested with a best-case and worst-case set of input values. For many classical sorting algorithms, the worst-case execution time occurs when the input set is reversed with respect to the direction of the $\mathrm{sort}()$ operation. Conversely, the best-case execution time will occur when the input set is already fully sorted. Each algorithm was tested using a sample dataset consisting of a fully sorted and a fully reversed input list, and the sorting algorithm execution time, t_{sort}, measured for each scenario. For the Xilinx Kintex-7 160T FPGA target, the execution time was measured by counting the number of clock ticks required for the sorting algorithm to execute, before converting to an execution time in microseconds using an FPGA clock frequency of 100 MHz. The execution time on the PXIe-8102 real-time CPU target was measured directly using the nanosecond resolution global timer provided by the timed loop block diagram construct in LabVIEW Real-Time. The execution time measured using this method is the total elapsed wall clock time taken for the sorting algorithm to execute to completion.

The results from the execution time range measurements for each sorting algorithm are shown in Figure 6.1(a) to (d), grouped according to the control hardware target platform and programming method. The maximum and minimum execution time measurements for the FPGA targets are fixed, since the sorting algorithm is implemented in fixed digital logic which is dedicated to the algorithm. The execution time ranges shown in Figure 6.1(a) for LabVIEW Standard running on the real-time CPU target are not fixed since the CPU is shared between other tasks. As a result, the execution time range shown is an estimate, however, is still representative of the typical order of execution time possible using LabVIEW Standard on the real-time CPU target.

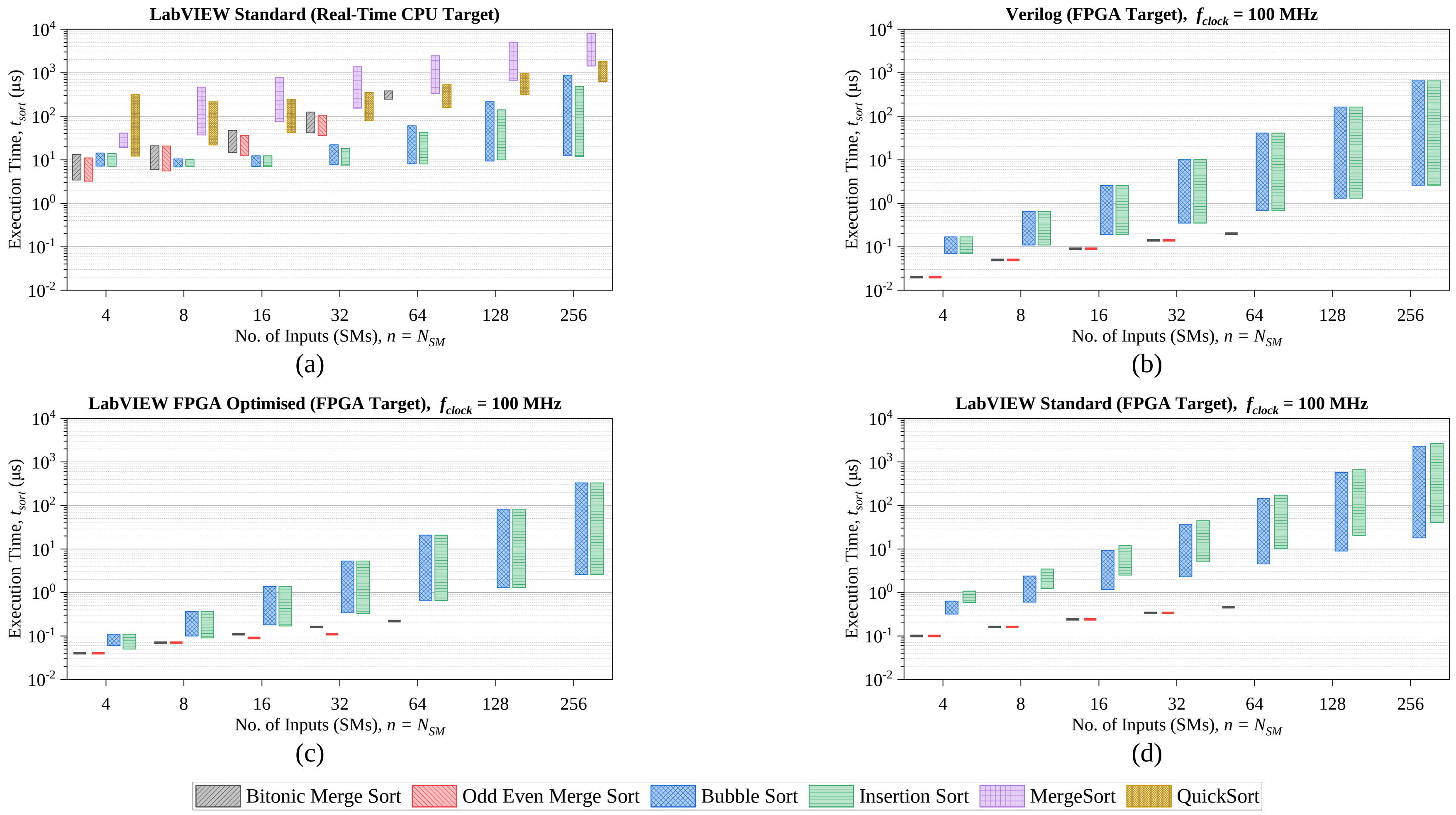


Figure 6.1: Execution time range as a function of number of inputs vs. sorting algorithm implemented in (a) LabVIEW Standard on PXIe-8102, (b) Verilog on FPGA, (c) LabVIEW FPGA Optimised on FPGA, and (d) LabVIEW Standard on FPGA

6.3.1 Real-Time CPU Target

Inspecting the results plotted for the real-time CPU target in Figure 6.1(a) first: bitonic merge sort and odd-even merge sort exhibit a range of execution times whilst the sorting networks running on the FPGA targets in Figure 6.1(b) to (d) have a fixed execution time. As outlined in Section 4.2, sorting networks are built from a fixed sequence of compare-swap operators. When implemented in fixed digital logic on an FPGA, each operator takes a fixed number of clock cycles to execute, leading to a fixed execution time for a given value of n. On the real-time CPU target however, the execution time for each operator will vary as the CPU is shared between other tasks, resulting in a non-fixed execution time.

Furthermore, the CPU target cannot execute compare-swap operators on the same stage of the sorting network in true parallel and instead forces sequential execution. This causes an increase in execution time range of 1 to 2 orders of magnitude over the same sorting network on all FPGA target implementations. As a result, both sorting networks offer sub-optimal performance on the real-time CPU target and another sorting algorithm should be chosen instead.

Comparing the classical sorting algorithms implemented on the real-time CPU target, bubble sort and insertion sort exhibit the lowest minimum and maximum execution times when compared to merge sort and quick sort. Both merge sort and quick sort use a divide-and-conquer approach to sorting. In this approach, the input list is divided into smaller sub-lists using recursion, before rebuilding the sub-lists in sorted order. In the LabVIEW Standard implementation of these algorithms, recursion is implemented using recursive calls to the same sub VI (virtual instrument, analogous to a function in a text-based programming language). Each sub VI call incurs a time delay as the CPU initialises new data structures for the sub VI and begins executing the sub VI code. This leads to an execution time overhead when compared to bubble sort and insertion sort on the same target, neither of which use recursion.

Based on the results in Figure 6.1(a), the execution time of the sorting algorithms implemented on the real-time CPU target was not evaluated further in this work, due to the non-determinism of sorting algorithm execution on this target. Furthermore, arm-level control loops (including CBC) in an MMC are typically implemented on an FPGA or digital signal processor (DSP) control hardware target, so the results from the FPGA target are of greater interest.

6.3.2 FPGA Target

The execution time range data for the three programming methods on the FPGA target are plotted in Figure 6.1(b) to (d). Considering the two sorting networks first, it can be seen that the execution time at each value of n is fixed, and that both bitonic merge sort and odd-even merge sort exhibit the lowest execution time out of the algorithms tested.

At each value of n, the execution times for the Verilog and LabVIEW FPGA Optimised implementations of the sorting networks are within +2 to −3 clock cycles of each other. When compared to LabVIEW Standard, LabVIEW FPGA is a lower-level abstraction which allows the programmer to implement digital logic with single clock cycle resolution. As a result, the LabVIEW FPGA Optimised implementation maps more directly to the underlying hardware description language and closely mirrors the Verilog implementation of the sorting networks. LabVIEW Standard is a higher-level abstraction which requires the insertion of additional logic by the LabVIEW FPGA Compilation Tool to preserve the LabVIEW dataflow execution paradigm. This additional logic incurs a clock cycle overhead and is the cause of the higher minimum and maximum execution times for all algorithms shown in Figure 6.1(d) when compared to LabVIEW FPGA Optimised and Verilog implementations.

For the two classical sorting algorithms (bubble sort and insertion sort), the LabVIEW FPGA Optimised implementation exhibits lower minimum and maximum execution times than the equivalent Verilog implementation. The maximum execution times of the Verilog implementations are between 1.5 to 2.0 times the maximum execution times of the LabVIEW FPGA Optimised implementation, across all values of n. This is due to differences in programming the finite-state machine (FSM) which controls algorithm execution. The Verilog implementation of both algorithms requires a ‘stall’ state when array values are swapped, which consumes 1 clock cycle, whilst the LabVIEW FPGA Optimised implementation does not require this.

Across all three FPGA targets, the minimum execution time of bubble sort and insertion sort grows linearly ($O(n)$), according to the best-case time complexity quoted in Table 4.1. The maximum execution time grows quadratically ($O(n^2)$), according to the worst-case time complexity quoted in Table 4.1. From the data presented in Figure 6.1(b) to (d), it can be concluded that implementing any of the sorting algorithms tested in this work in LabVIEW FPGA Optimised provides a good balance between execution time and implementation effort.

6.4 Comparison of Balancing Method Execution Time Window and Sorting Algorithm Execution Time

As outlined in the review of CBC methods in Section 3.2, the chosen CBC method dictates the time window available for the sorting algorithm to execute and return a sorted list. Alternatively, the time window can be dictated by the fundamental control cycle sampling period during which capacitor voltages are acquired, the CBC loop is triggered, and updated firing signals are communicated to the SMs.

In addition to dictating the time available for sorting algorithm execution, the selected CBC method will affect the existing ordering of SM capacitor voltages at the input to the sorting algorithm. As stated in Section 4.1.2, for bubble sort and insertion sort, the existing ordering of the input set will directly influence the sorting algorithm execution time, t_{sort}, whilst in the case of merge sort and quick sort the existing ordering has no effect.

When the sorting algorithm execution time exceeds the time window available for execution, this will lead to a delay in updating SM firing signals for balancing purposes. This may cause an increase in the SM capacitor ripple voltage, which will reduce the lifespan of the SM capacitor due to increased ripple current [90] or will cause an imbalance between individual SM capacitor voltages, leading to uneven voltage steps in the AC output waveform.

In this section, the time window available for the sorting algorithm to execute is compared with the real-world execution time of the FPGA-based sorting algorithm implementations listed previously in Table 6.3. The PSCAD/EMTDC MMC model introduced in Chapter 5 was used to generate synthetic capacitor voltage data for three CBC methods:

- Periodic CBC operating at f_{s-CBC} = 1 kHz
- Average tolerance band (ATB) with a tolerance band of ± 5 % around the instantaneous average SM capacitor voltage, $\overline{V_{cap}}(t)$
- Cell tolerance band (CTB) with a tolerance band of ± 0.091 kV around the nominal SM capacitor voltage, $V_{cap-nom}$

The SM capacitor voltages for a single converter arm were then sorted offline and the sorting algorithm execution time measured. The percentage of CBC loop sampling instants where the sorting algorithm execution time exceeded a maximum allowable delay was then plotted. This allows for a thorough evaluation of the performance each sorting algorithm implementation when used with different CBC methods and converter operating scenarios.

6.4.1 Terminology

The relationship between control cycle sampling period, $T_{s-control}$, CBC method sampling period, T_{s-CBC}, and sorting algorithm execution time, t_{sort}, is shown in Figure 6.2. Time is shown on the *x*-axis in units of sampling instants, k, since this work is focussed upon CBC methods implemented in digital control hardware which operates in the discrete time domain. Where CBC is implemented in hardware operating in hard real-time, $T_{s-control}$ is fixed and is the timebase from which all other control loop sampling periods are derived. In the case of CBC, during each control cycle three key operations are performed:

- Capacitor voltage measurements are acquired from the SMs.
- Capacitor balancing is triggered dependent upon the chosen CBC method and/or measured SM capacitor voltages.
- Updated firing signals are communicated to the SMs.

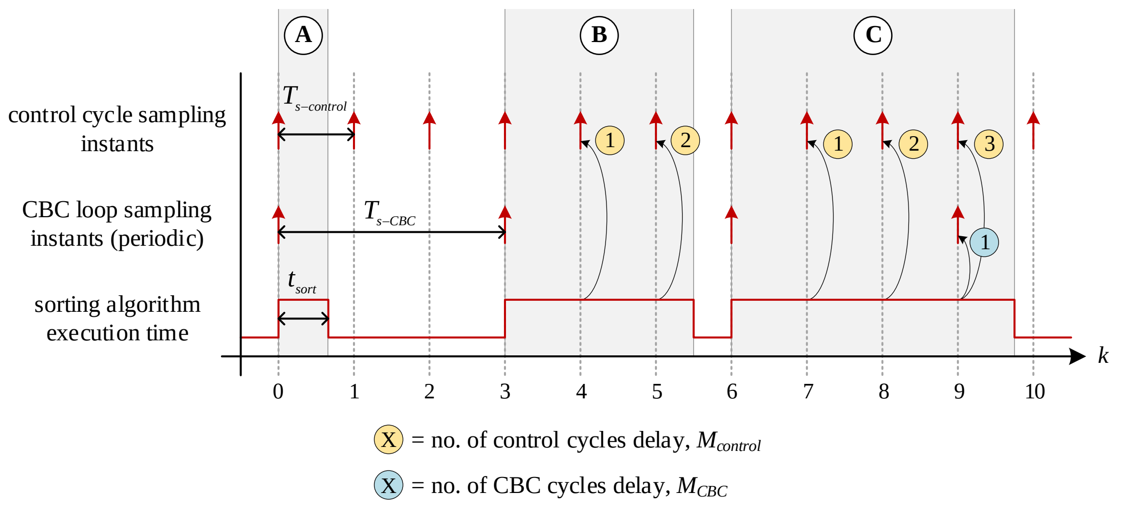

A
B
C
control cycle sampling instants
CBC loop sampling instants (periodic)
sorting algorithm execution time
$T_{s-control}$
T_{s-CBC}
t_{sort}
1
2
1
2
3
1
k
0
1
2
3
4
5
6
7
8
9
10
X = no. of control cycles delay, $M_{control}$
X = no. of CBC cycles delay, M_{CBC}

Figure 6.2: Timing diagram showing the relationship between $T_{s-control}$, T_{s-CBC}, t_{sort}, $M_{control}$, M_{CBC}, against sampling instant number, k

In this work, $T_{s-control}$ is equal to the PSCAD/EMTDC simulation timestep. The timestep was set to 50 µs to provide a balance between simulation fidelity and runtime. Limited information is available in the public domain regarding the base control cycle sampling period used in industrial HVDC MMCs. The book, “Design, Control, and Application of Modular Multilevel Converters” [18], whose authors include recognised researchers working at ABB, quote a typical control cycle sampling frequency of 10 kHz ($T_{s-control}$ = 100 µs).

Furthermore, the paper, “Modelling and Current Control of Modular Multilevel Converters Considering Actuator and Sensor Delays” [91], whose authors include Siemens employees, quotes a control cycle sampling frequency of 25 kHz ($T_{s-control}$ = 40 μs). The selection of a simulation timestep of 50 μs for this work is justified based on this information.

In Figure 6.2, the sampling points shown are for the case where periodic CBC is chosen as the CBC method. In this case, T_{s-CBC} is fixed at an integer multiple of $T_{s-control}$. For aperiodic CBC methods such as ATB and CTB, T_{s-CBC} varies from one sampling instant to the next, since the CBC loop is only triggered when a SM capacitor voltage exceeds a pre-defined tolerance band.

As can be seen in Figure 6.2, the sorting algorithm is triggered at each CBC loop sampling instant and executes asynchronously; that is, the sorting algorithm may finish executing at a time which is not aligned to a control cycle sampling instant. The sorting algorithm may finish executing:

- Before the next control cycle sampling instant (as shown in region **A** in Figure 6.2). In this scenario, the sorting algorithm execution time will not affect CBC performance, since sorting is completed before the next control cycle.
- After a delay of one or more control cycles, but before the next CBC loop sampling instant (region **B**). In this scenario, the sorting algorithm execution time may affect CBC performance, since updating the firing signals to the SMs for balancing purposes is delayed by one or more control cycles.
- After a delay of one or more control cycles and one or more CBC loop cycles (region **C**). In this scenario, the sorting algorithm execution time will affect CBC performance, since firing signals are not updated before the next CBC loop cycle.

If a sorting algorithm with a non-deterministic execution time is used, the execution time will vary from one CBC loop sampling instant to the next and may fall into region **A**, **B**, or **C**. Two variables, $M_{control}$ and M_{CBC}, are defined in this work and are used to measure the integer number of samples delay introduced by the sorting algorithm. Only integer values are of interest, since sorting algorithm execution time delays which fall between control cycle sampling instants cannot be resolved by the CBC loop.

$M_{control}$ counts the number of samples delayed relative to the control cycle sampling instants when $t_{sort} \geq T_{s-control}$. M_{CBC} counts the number of samples delayed relative to the CBC loop sampling instants when $t_{sort} \geq T_{s-CBC}$. $M_{control}$ and M_{CBC} and are shown in the yellow- and blue-coloured circles in Figure 6.2, and are calculated for a given sampling instant, k, according to Equation 6.1 and Equation 6.2.

$$M_{control}[k] = \text{floor}\left(\frac{t_{sort}[k]}{T_{s-control}}\right) \qquad 6.1$$

$$M_{CBC}[k] = \text{floor}\left(\frac{t_{sort}[k]}{T_{s-CBC}[k]}\right) \qquad 6.2$$

6.4.2 Selection of Maximum Allowable Sample Delay

Whilst researching the effects of internal converter communication network delays, the authors in [92] found that the maximum capacitor voltage deviation remained below 1 % of the nominal capacitor voltage, $V_{cap-nom}$, for a 2-sample delay ($M_{control}$ = 2). As the number of samples delay was increased further, the maximum capacitor voltage deviation increased, up to a maximum of 2 % for an 8-sample delay ($M_{control}$ = 8). A control cycle sampling period of 100 μs was used, and $V_{cap-nom}$ was set at 170 V. The results in [92] demonstrate that the CBC loop can tolerate some delay whilst maintaining SM capacitor voltages within an acceptable tolerance band around $V_{cap-nom}$.

Motivated by the research in [92], a maximum allowable control cycle sample delay, $M_{control-threshold}$, was determined empirically by simulation for use in this work. Applying a threshold to $M_{control}[k]$ ensures that only sampling instants where $M_{control}[k]$ is large enough to cause an unacceptably large capacitor voltage deviation are counted. $M_{control-threshold}$ was chosen such that where $M_{control}[k] < M_{control-threshold}$, the maximum SM capacitor voltage deviation from $V_{cap-nom}$ would remain within approximately ±8 % of $V_{cap-nom}$. This resulted in a value of $M_{control-threshold}$ = 8 being chosen, corresponding to a 400 μs delay where $T_{s-control}$ = 50 μs. The selection process for $M_{control-threshold}$ is described in more detail in Appendix C.

6.4.3 Execution Time Results

To compare the time available for the sorting algorithm to execute with the real-world execution time of the FPGA-based sorting algorithms, $T_{s-CBC}[k]$ was calculated at each CBC loop sampling instant for the three CBC methods (periodic, ATB, and CTB). Three converter operating scenarios were tested. To simplify the analysis, only the results from the upper arm of phase A are shown, since the other five arms will exhibit similar results. $M_{control}[k]$ was then calculated using the sorting algorithm execution time measurements for the six sorting algorithm implementation combinations listed in Table 6.4.

Sorting Algorithm	Programming Methods
Bubble Sort	Verilog LabVIEW FPGA Standard LabVIEW FPGA Optimised
Insertion Sort	Verilog LabVIEW FPGA Standard LabVIEW FPGA Optimised

Table 6.4: List of sorting algorithms and programming method combinations tested

The three CBC methods tested are each triggered a different number of times within a given simulation time frame. To allow a direct comparison between CBC methods, the number of times $M_{control}[k]$ is greater than or equal to $M_{control-thresh}$ was converted to a percentage of the total number of CBC loop sampling instants within the selected time frame. This process is shown in Figure 6.3.

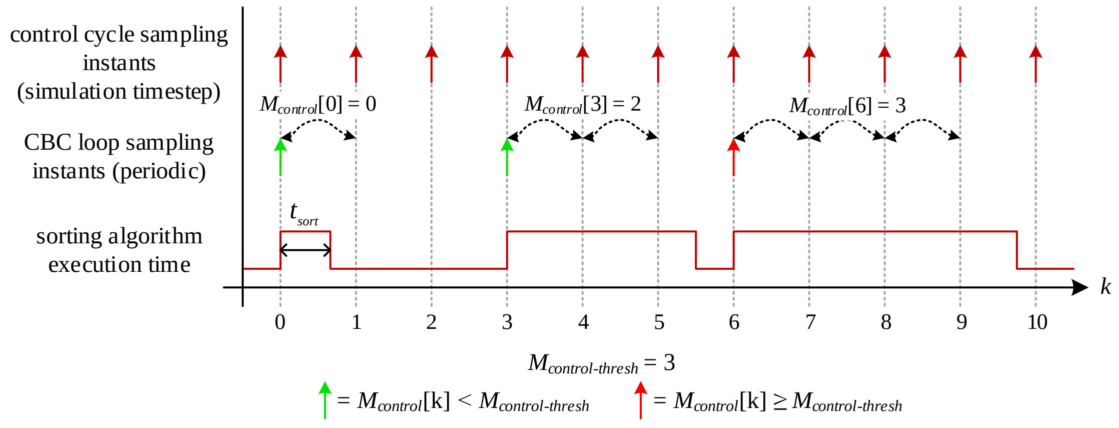

control cycle sampling instants (simulation timestep)
CBC loop sampling instants (periodic)
sorting algorithm execution time
$M_{control}[0] = 0$
$M_{control}[3] = 2$
$M_{control}[6] = 3$
t_{sort}
0
1
2
3
4
5
6
7
8
9
10
k
$M_{control-thresh} = 3$
$= M_{control}[k] < M_{control-thresh}$
$= M_{control}[k] \geq M_{control-thresh}$

Figure 6.3: Timing diagram showing the relationship between $M_{control}[k]$, $M_{control-thresh}$, and total number of CBC loop sampling instants

For example, in Figure 6.3 the simulation time frame spans from k = 0 to k = 10. The CBC loop is triggered three times within this time frame and $M_{control}[k]$ is greater than or equal to $M_{control-thresh}$ at one CBC loop sampling instant when k = 6. This corresponds to 33 % of the total number of CBC loop sampling instants being greater than or equal to $M_{control-thresh}$ = 3.

Supplementary plots showing CBC loop sampling instants against arm current, SM insertion index, and SM capacitor voltage are included in Appendix D. These show the variation in CBC loop sampling period between the three CBC methods and three simulation scenarios tested.

Scenario 1: Steady State Power Import

A steady state operating scenario was tested first, with the MMC operating in inverter mode configured to import 1200 MW of active power (P^*) and 0 MVAr of reactive power (Q^*). This provides a base case for comparison with the other operating scenarios. For each operating scenario a simulation time frame of t = 2.05 seconds to t = 2.35 seconds was analysed. The converter-level response for scenario 1 is plotted in Figure 6.4. The percentage of CBC loop cycles where the sorting algorithm execution time exceeds the threshold is shown in the bar graph in Figure 6.5 for the three CBC methods and six sorting algorithm implementations.

As can be seen in Figure 6.5, where periodic CBC is used, the 8-cycle threshold is exceeded at all CBC loop sampling instances by all sorting algorithm implementations, apart from insertion sort implemented in LabVIEW FPGA Optimised. Similar results are observed when ATB is used. From these results it can be deduced that the list of SM capacitor voltages at the input to the sorting algorithm exhibits poor pre-sortedness. As a result, more compare-swap operations (and therefore more clock cycles) are required to generate a sorted list.

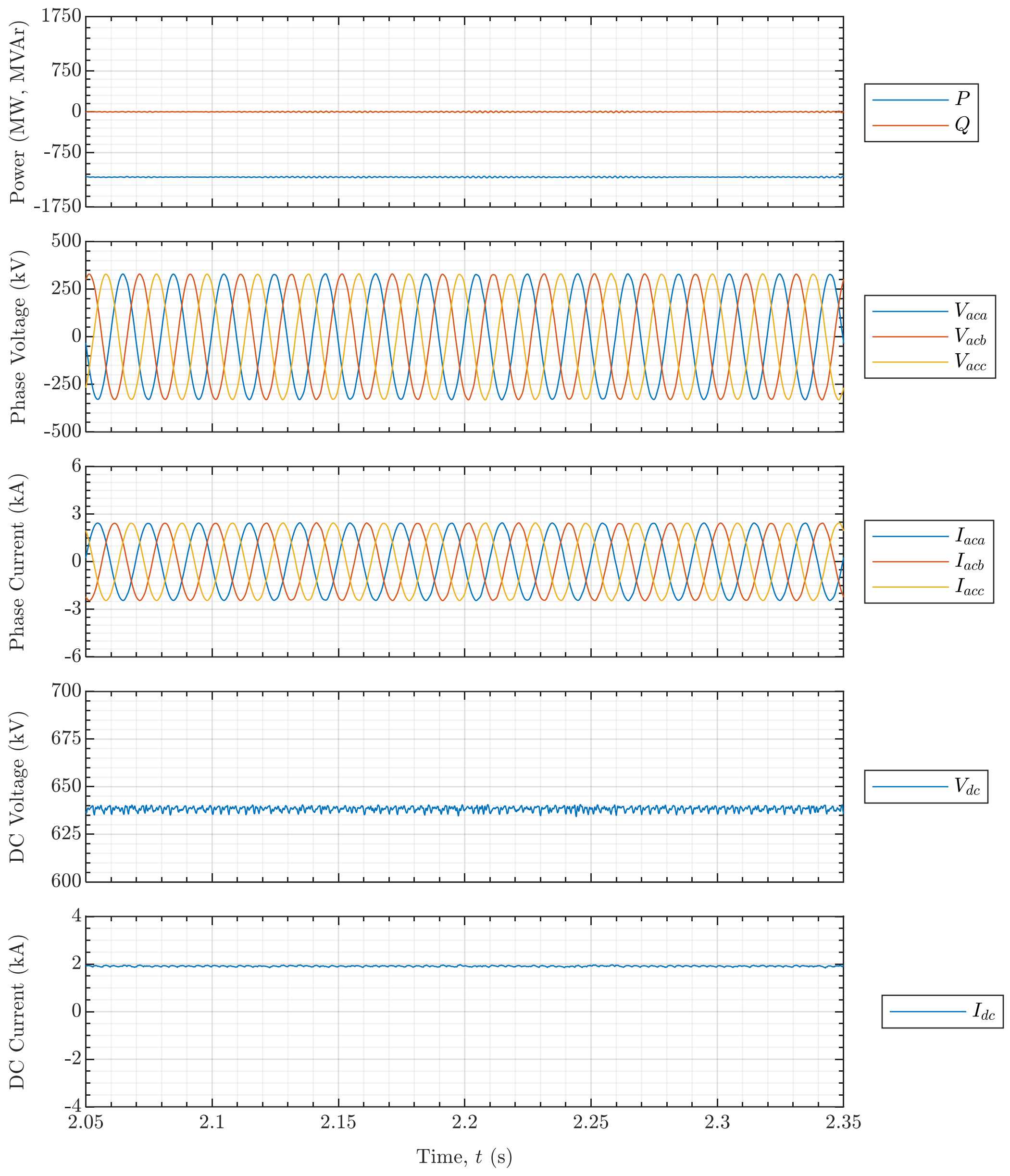


Figure 6.4: Converter-level response for scenario 1, steady state operation at $P^* = -1200$ MW, $Q^* = 0$ MVAr (import, inverter mode)

In the case of CTB, the 8-cycle threshold is exceeded at a smaller percentage of CBC loop sampling instances across all sorting algorithm implementations. When compared with periodic and ATB methods, capacitor balancing is triggered at a higher frequency when CTB is used. Table 6.5 lists the total number of CBC loop sampling instances for each CBC method in the simulation time frame under consideration. This shows that CTB has a higher number of trigger instances than either periodic or ATB methods.

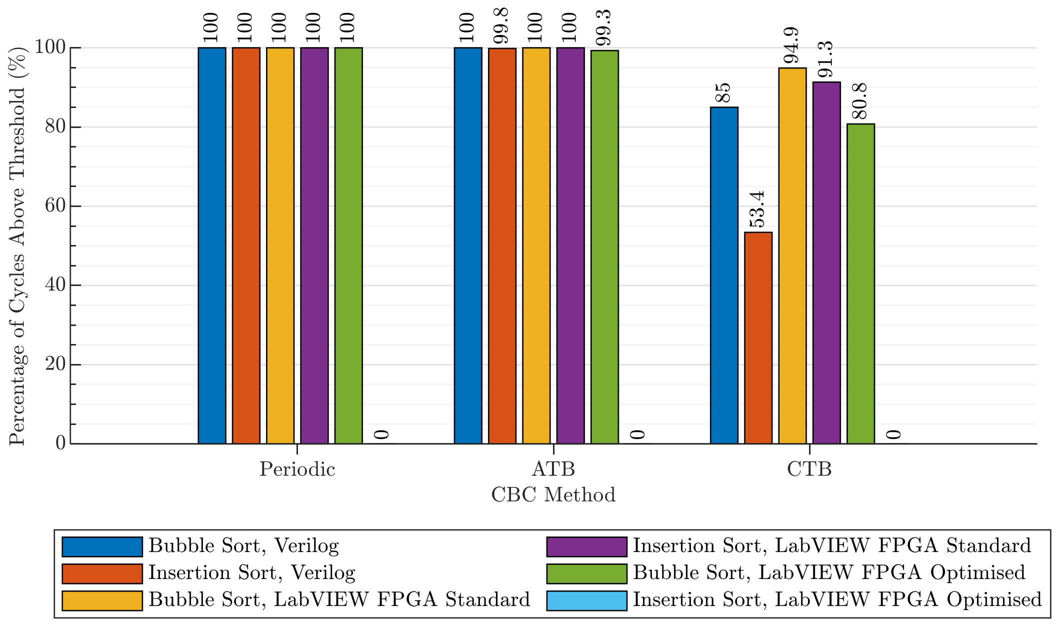


Figure 6.5: Percentage of CBC loop cycles where $M_{control-thresh}$ = 8 was exceeded for scenario 1

CBC Method	**Total Number of CBC Loop Sampling Instances** (t = 2.05 to t = 2.35 seconds)
Periodic	301
ATB	2469
CTB	4701

Table 6.5: Total number of CBC loop sampling instances per CBC method, scenario 1

The higher sampling frequency of CTB when compared with periodic and ATB methods leads to a smaller deviation in individual SM capacitor voltages from $V_{cap-nom}$ in the interval between CBC loop sampling instances. As a result, the list of SM capacitor voltages at the input to the sorting algorithm exhibits better pre-sortedness, therefore fewer clock cycles are required generate a sorted list.

Scenario 2: Power Flow Reversal

In the second scenario, the MMC was configured to operate in inverter mode importing 1200 MW of active power and 0 MVAr of reactive power. A full active power flow reversal was then initiated starting at t = 2.1 seconds and finishing at t = 2.35 seconds. The converter-level response for scenario 2 is shown in Figure 6.6 and the percentage of cycles where the sorting algorithm execution time exceeds the threshold is shown in Figure 6.7.

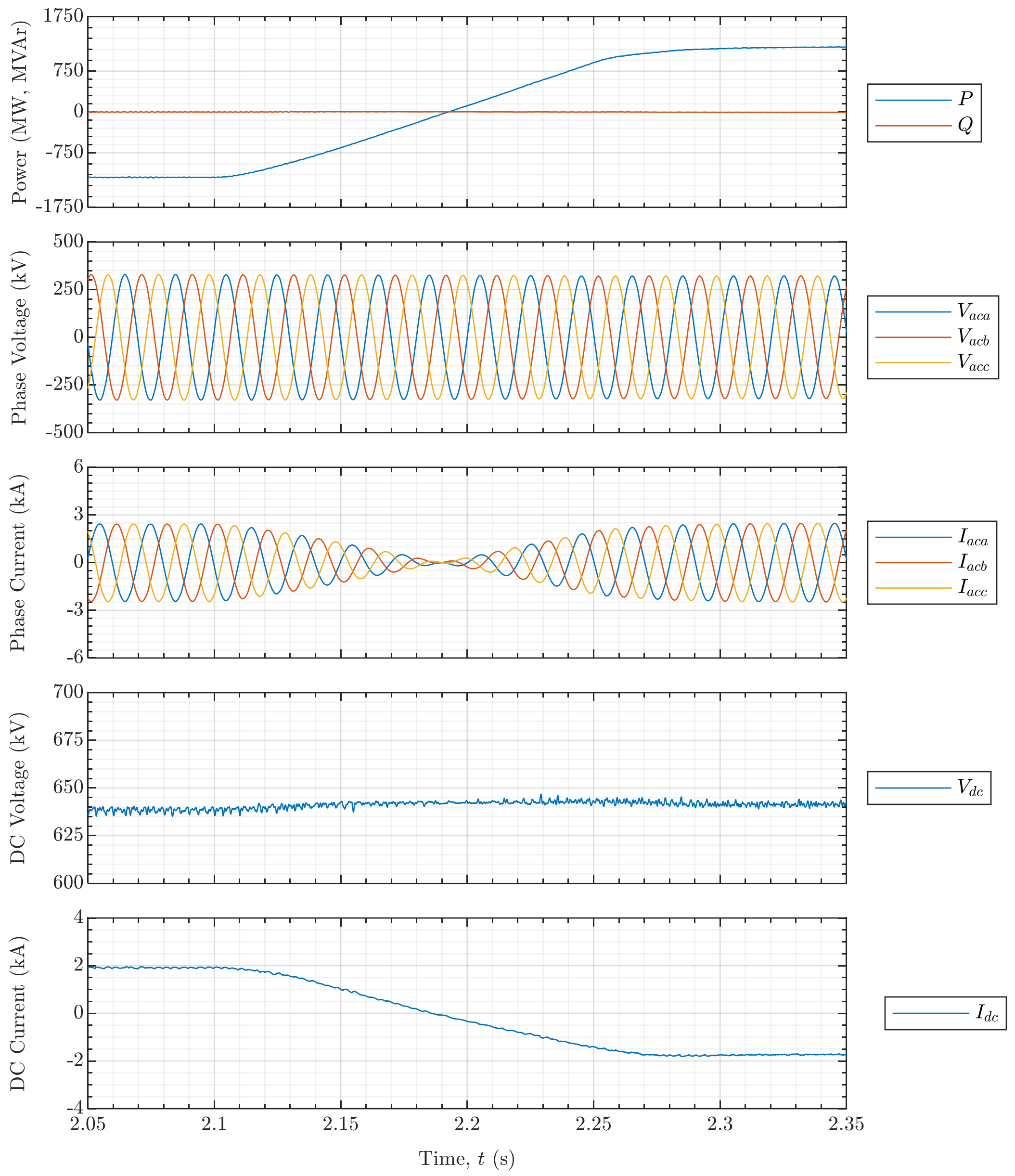


Figure 6.6: Converter-level response for scenario 2, full power flow reversal at approx. $t = 2.1$ seconds from $P^* = -1200$ MW (import, inverter), $Q^* = 0$ MVAr, to $P^* = 1200$ MW, $Q^* = 0$ MVAr (export, rectifier)

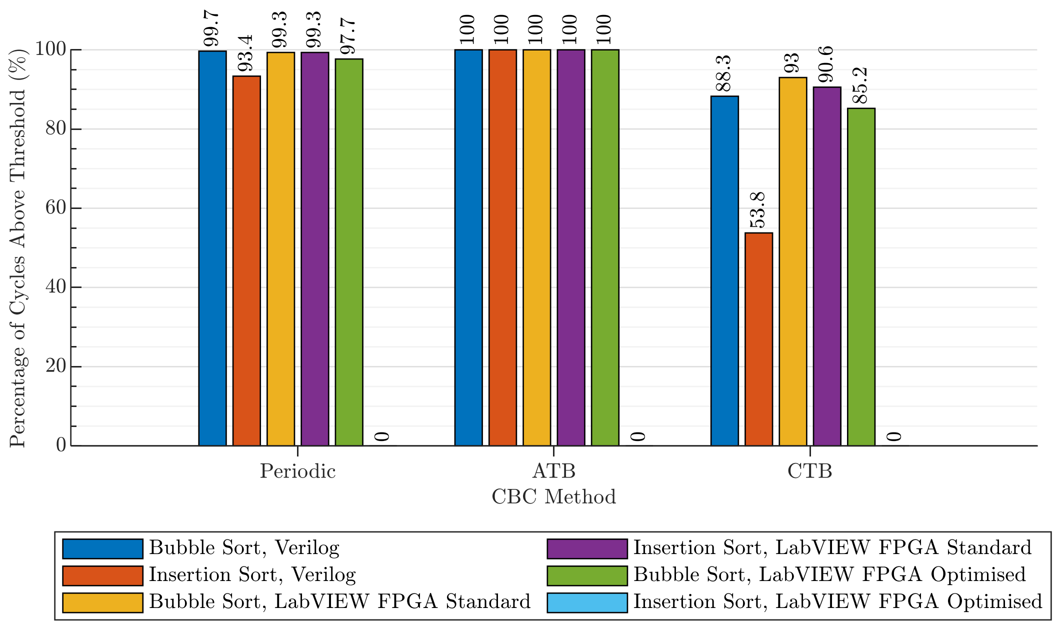


Figure 6.7: Percentage of CBC loop cycles where $M_{control-thresh}$ = 8 was exceeded for scenario 2

Comparing data for periodic CBC in Figure 6.7 with the data for scenario 1 in Figure 6.5, the percentage of cycles above the threshold is lower for all sorting algorithm implementations. This can be explained by inspecting the converter output current in Figure 6.6. During power flow reversal, the converter output current, $I_{ac(abc)}$, drops to zero as the active power is decreased, before increasing as the active power ramps up again. Correspondingly, the current in each converter arm will also decrease to zero before increasing again during this interval, as shown in Figure 6.8 for the upper arm of phase A.

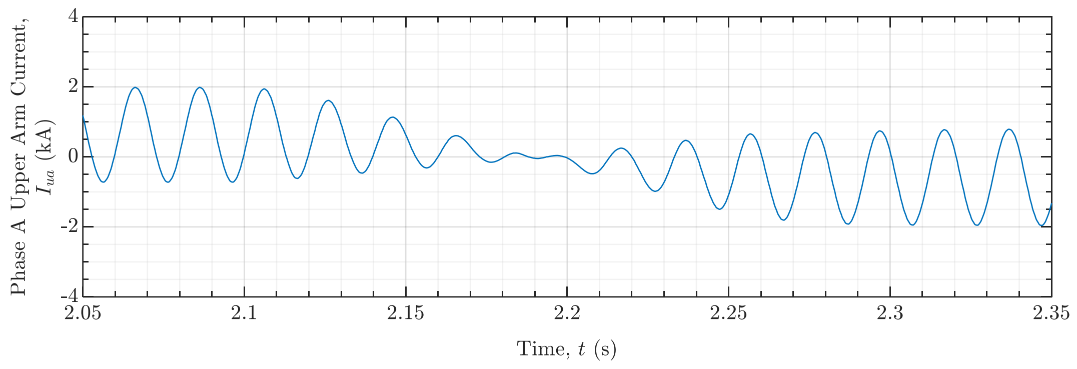


Figure 6.8: Phase A, upper arm current for scenario 2

Recalling the SM capacitor voltage dynamics described in Section 2.1, inserted SMs are charged or discharged dependent upon the arm current direction, whilst the rate of charge or discharge is determined by the arm current magnitude. In Figure 6.8, the arm current magnitude is less than the steady state value from t = 2.1 seconds to t = 2.3 seconds. In this region, SM capacitors will be charged or discharged at a slower rate and SM capacitor voltages will diverge less between each CBC loop sampling instant. As a result, the list of SM capacitor voltages will be more ordered prior to sorting and fewer compare-swap operations are required to generate the sorted output list. This leads to a reduction in sorting algorithm execution time and the percentage of CBC loop cycles which exceed the delay threshold.

The total number of CBC loop sampling instances for scenario 2 are listed in Table 6.6. Comparing the values for CTB with scenario 1 and 3, the total number of sampling instances is lower, since the rate of charging and discharging of SM capacitors is reduced during power flow reversal and capacitor voltages remain within the tolerance band.

CBC Method	**Total Number of CBC Loop Sampling Instances** (t = 2.05 to t = 2.35 seconds)
Periodic	301
ATB	1498
CTB	3917

Table 6.6: Total number of CBC loop sampling instances per CBC method, scenario 2

Despite the reduction in number of CBC loop sampling instances, both ATB and CTB methods exhibit a higher percentage of cycles above the threshold across all sorting algorithm implementations compared with scenario 1 and 3. This may be due to a reduction in the CBC loop sampling period and therefore the time available for the sorting algorithm to execute. Alternatively, the increase in cycles above the threshold may be due to a more unordered list of SM capacitor voltages at the input to the sorting algorithm which will take longer to sort.

Scenario 3: AC 3 Phase-ground Fault

In the third scenario, the MMC was configured to operate in inverter mode, importing 1200 MW of active power and 0 MVAr of reactive power. An instantaneous symmetric 3 phase-ground short circuit (0 ohm) fault was then applied at the point of common coupling (PCC) at t = 2.1 seconds and cleared after 140 milliseconds, within one simulation timestep (50 μs). Control and protection functions were not initiated during the fault and the converter remained unblocked and continued to feed current into the fault. The fault location is shown in the simulation model circuit diagram in Figure 5.1 in Chapter 5. The converter-level response for is shown in Figure 6.9 and the percentage of cycles where the sorting algorithm execution time exceeds the threshold is shown in Figure 6.10.

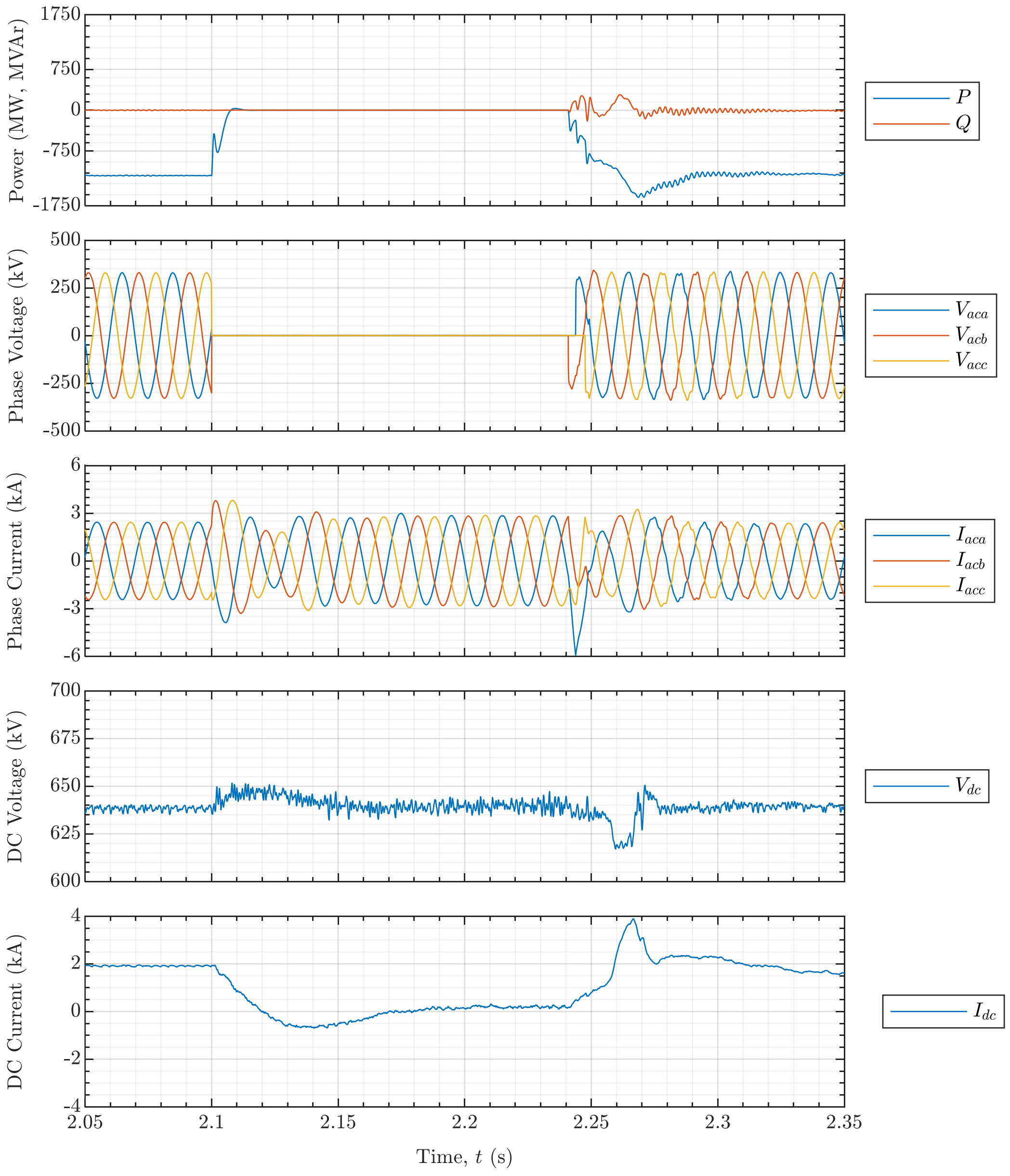


Figure 6.9: Converter-level response for scenario 3, 3-phase symmetric AC phase-ground fault at PCC at t = 2.1 seconds for 140 milliseconds

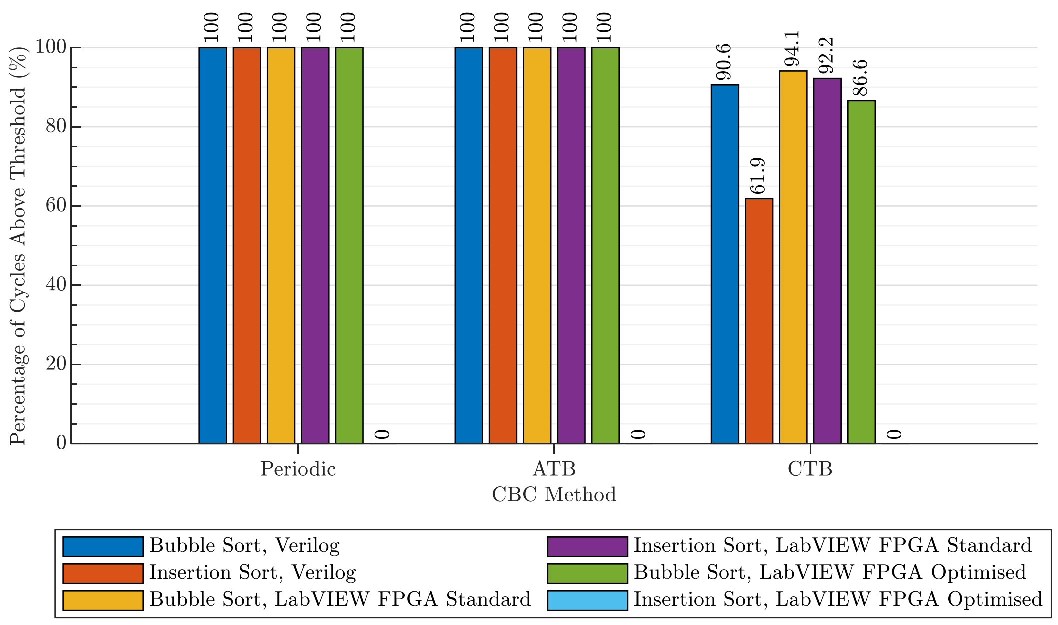


Figure 6.10: Percentage of CBC loop cycles where $M_{control-thresh} = 8$ was exceeded for scenario 3

From Figure 6.10, all sorting algorithm implementations, apart from insertion sort implemented in LabVIEW FPGA Optimised, fail to finish executing within the 8-cycle threshold, where periodic or ATB methods are used. In addition, the percentage of cycles above the threshold is higher across all implementations for the CTB method, when compared with scenarios 1 and 2. Across all CBC methods, these results can be attributed to the interaction between arm current and SM insertion index during the fault event.

Observing the arm current waveform in Figure 6.11, prior to the fault occurring at $t = 2.1$ seconds, the steady state arm current magnitude is in the range of approximately −0.75 kA to 2 kA. In the cycle immediately following the fault, the arm current peaks at a maximum of 2.75 kA and a minimum of −1.25 kA, before stabilising whilst the fault remains present. Upon clearing the fault at $t = 2.24$ seconds, the arm current peaks at 4 kA and stabilises to the steady state magnitude within two cycles.

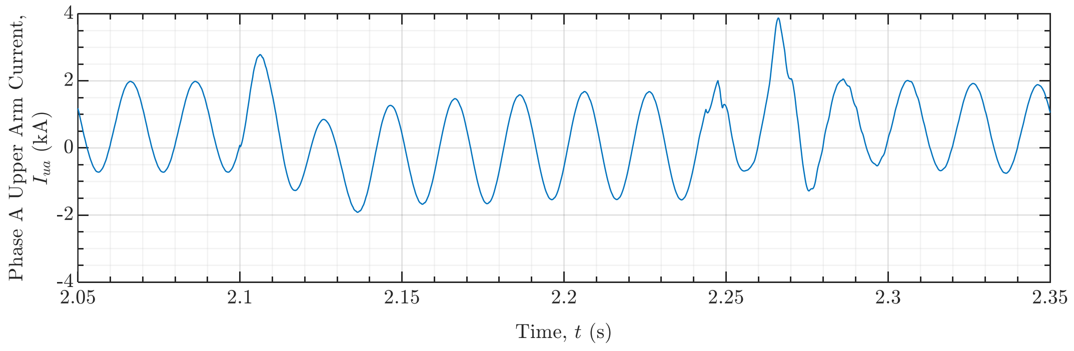


Figure 6.11: Phase A, upper arm current for scenario 3

In addition to the large arm current disturbance, the range of the SM insertion index (N_{on}) is also reduced during the fault event, as shown in Figure 6.12. A reduced number of inserted SMs, coupled with an increase in arm current magnitude will cause the inserted SMs to charge or discharge at a faster rate, causing a larger deviation in SM capacitor voltage between CBC loop sampling instants. This will lead to a less ordered list of SM capacitor voltages at the input to the sorting algorithm, which will correspondingly cause an increase in sorting algorithm execution time.

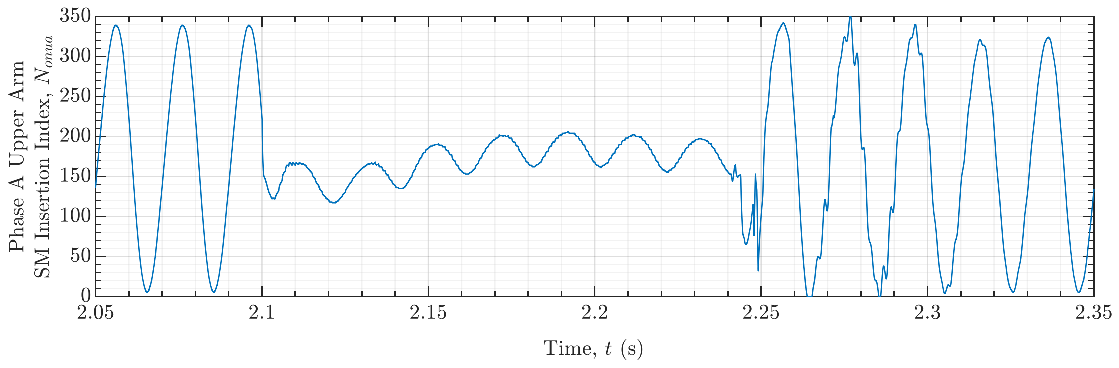


Figure 6.12: Phase A, upper arm SM insertion index for scenario 3

The total number of CBC loop sampling instances for the three CBC methods are listed in Table 6.7. Compared with scenarios 1 and 2, the number of CBC loop firings for CTB is higher, due to CTB using a fixed upper and lower SM capacitor voltage limit which does not track the deviation in instantaneous mean SM capacitor voltage ($\overline{V_{cap}}(t)$) during the fault event. Conversely, ATB is triggered fewer times than scenarios 1 and 2. In the ATB method, the upper and lower voltage limits are calculated using $\overline{V_{cap}}(t)$ at each control cycle sampling instant, therefore the voltage limits track the deviation in $\overline{V_{cap}}(t)$ which occurs during the fault event.

CBC Method	**Total Number of CBC Loop Sampling Instances** (t = 2.05 to t = 2.35 seconds)
Periodic	301
ATB	1424
CTB	4879

Table 6.7: Total number of CBC loop sampling instances per CBC method, scenario 3

Summary

From the results presented in this section, it can be seen that the choice of sorting algorithm and implementation method has a direct effect on whether or not the sorting algorithm finishes executing within the required time window. Across all CBC methods and operating scenarios, insertion sort has a consistently lower execution time than bubble sort, when comparing for the same implementation method. Insertion sort can be recommended as the preferred sorting algorithm where a classical, sequential sorting algorithm is used.

Of note is insertion sort implemented in LabVIEW FPGA Optimised, which has a delay of less than 8 control cycles across all CBC methods and operating scenarios. As outlined in Section 6.3.2, insertion sort in the form implemented in this work does not require an additional ‘stall’ cycle when a swap operation is required, which reduces execution time when compared to LabVIEW FPGA Standard and Verilog implementations.

Across all three operating scenarios, all other sorting algorithms exhibited percentage of cycles above the 8-cycle threshold which was equal to or approaching 100 %, when periodic or ATB methods were chosen. To remedy this, it may be necessary to increase the FPGA clock frequency above 100 MHz to ensure that the sorting algorithm executes in the available time window.

Finally, across all scenarios, the CTB capacitor balancing method consistently has the highest number of sampling instances. As the CBC loop is triggered at a higher frequency, SM capacitor voltages diverge less between sampling instances, and the sorting algorithm is presented with a better-ordered input list, reducing the sorting algorithm execution time.

Histogram plots of the number of missed control cycles for each sorting algorithm, CBC method, and operating scenario are included in Appendix D. These provide a more detailed insight into number of control cycles delay by showing the distribution of missed control cycles in histogram form, rather than applying a simple threshold value as has been done in this chapter.

The analysis presented in this section can be extended further by incorporating the real-world sorting algorithm execution delay into the CBC loop used in the simulation model. This will allow evaluation of the effect of the execution time delay upon the capacitor voltage ripple, including the non-deterministic behaviour of the sorting algorithm from one CBC loop sampling instant to the next.

6.5 Sorting Algorithm FPGA Logic Resource Usage

As outlined in the introduction to this chapter, another objective of this research is to measure FPGA logic resources used by each sorting algorithm when implemented using different programming methods. The results from this work can be used to guide the choice of sorting algorithm for sorting-based CBC methods when the CBC loop is implemented alongside other control functions on the same FPGA.

To allow a comparison of the logic resource usage of bubble sort, insertion sort, bitonic merge sort, and odd-even merge sort, each algorithm was programmed using the three implementation methods. The number of inputs to each algorithm was then varied from $n = 4$ to $n = 256$ and the resulting implementation synthesised for the Xilinx Kintex-7 160T FPGA used on the NI PXIe-7857R. The number of logic slices used was then recorded and used as a measure of FPGA logic resource usage.

6.5.1 FPGA Synthesis Process

In FPGA firmware development, the term ‘synthesis’ is used to describe the process of translating an FPGA design from source code to a design which can be loaded onto an FPGA. It is analogous to the compilation process in software development, however the resulting design is implemented in fixed digital logic, rather than a set of instructions for execution on a processor or microcontroller. The synthesis process has several steps which map a design to the underlying logic blocks on the FPGA. These are shown in the flowchart in Figure 6.13.

As outlined in Section 6.2.2, LabVIEW and Verilog were used to implement the sorting algorithms tested in this work. The algorithm implementation step is shown at the top of the flowchart in Figure 6.13, the output of which is a set of Verilog source code files, or a LabVIEW VI. The next steps in the synthesis process are different for Verilog and LabVIEW implementations and are explained in the following sub-sections.

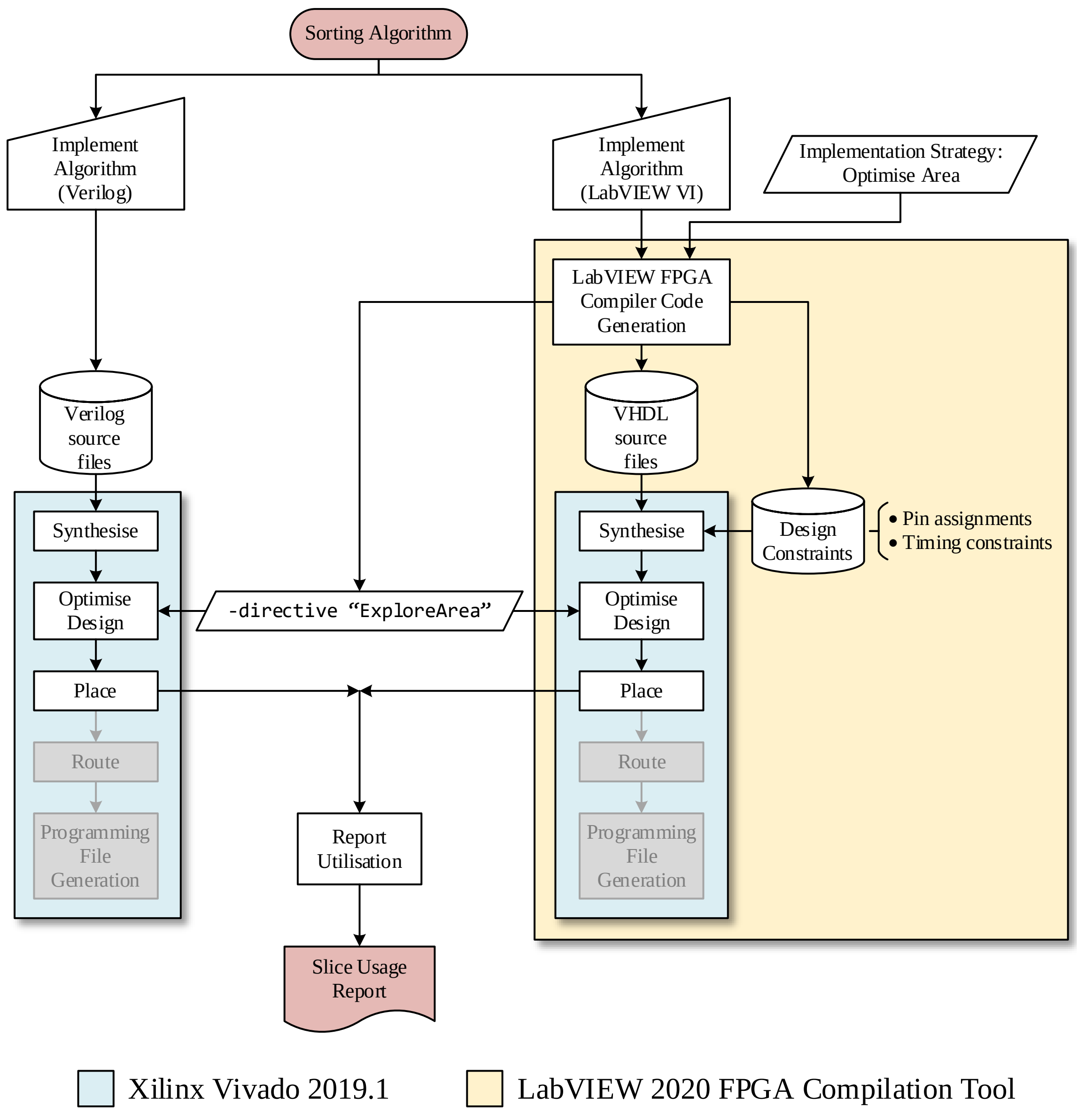


Figure 6.13: Synthesis process for sorting algorithm logic resource usage measurements

LabVIEW FPGA Compiler Code Generation

When using LabVIEW to program the Xilinx Kintex-7 in the PXIe-7857R, the high-level abstraction LabVIEW VI must first be translated into low-level synthesisable VHDL source code. This process is carried out by the LabVIEW FPGA Compilation Tool, shown by the yellow bounding box in Figure 6.13. In addition to VHDL source code generation, the compilation tool inserts FPGA target-specific VHDL modules and constraint files to support the integration of the Xilinx Kintex-7 on the PXIe-7857R. It also controls the overall compilation process and automatically invokes the Xilinx Vivado 2019.1 toolchain with the appropriate directives for the FPGA synthesis steps, as shown by the blue bounding box in Figure 6.13.

Synthesise

In the synthesis step of the compilation process, the compiler translates the HDL source code into digital logic elements such as logic gates and flip-flops, and a list of connections between these elements (netlist). By analysing the LabVIEW FPGA Compilation Tool log files, it was possible to extract the directives used by compilation tool when invoking the Xilinx Vivado toolchain. The same directives were then used when manually compiling the Verilog implementations of the sorting algorithms to provide a like-for-like comparison of logic resource usage.

Optimise Design

At each stage of the synthesis process, the compilation tool can be provided with directives to optimise the FPGA design. These may include optimising for area (minimise logic resource usage), timing (maximise clock frequency), or power (minimise power consumption). Since logic resource usage is the focus of this work, the LabVIEW FPGA Compilation Tool was configured to use the ‘Optimise Area’ implementation strategy. This corresponds to invoking the Vivado `opt_design` command with the `“ExploreArea”` directive. The same directive was therefore also used for manual compilation of the Verilog implementations of the sorting algorithms. These are shown in the parallelogram boxes in Figure 6.13.

Map

The mapping stage of the synthesis process translates logic elements from the optimised design onto the logic blocks provided by the underlying architecture of the FPGA, creating a network of logic blocks. At this stage the logic blocks are not assigned to specific locations on the FPGA. In the case of Xilinx FPGAs such as the Kintex-7 used in this work, logic blocks are referred to as ‘slices’ and contain four lookup tables and eight flip-flops [86]. FPGAs from other manufacturers use different terminology and architectures.

Place and Route

The final stage of the synthesis process, prior to generating a programming file for the FPGA, is to place and route the design. In the placement step, the compiler assigns the logic blocks from the mapping step to specific locations on the FPGA. The interconnections between the logic blocks are then defined in the routing step using the netlist generated previously. For the purposes of this work, the synthesis process was stopped following the placement step, since the required logic resource usage data could be extracted at this point.

Report Usage

Reports can be generated at each stage of the synthesis process to provide detailed information on the implementation of the design on the FPGA. Logic resource usage can be reported following completion of any of the steps outlined previously. For this work, logic resource usage post-placement is of interest, since this reports the total number of slices used when the sorting algorithm is implemented on the FPGA.

As stated previously, the LabVIEW FPGA Compilation Tool inserts additional support logic around the LabVIEW VI. As a result, the logic resource usage reported by the compilation tool is not specific to the sorting algorithm VI and includes the logic resources used by the support logic. To provide a direct comparison of sorting algorithm logic resource usage across all three programming methods, the logic resource usage was extracted from the LabVIEW FPGA synthesis process.

6.5.2 Logic Resource Usage Results

The total number of slices used by the design, N_{slice}, reported by the synthesis tool for each sorting algorithm and programming method combination at each value of N are plotted in Figure 6.14 and Figure 6.15. The graphs in both figures have been generated using the same dataset however are plotted with different subplots to allow for a range of comparisons to be carried out. The plots in Figure 6.14(a) to (d) show the total number of slices used by each sorting algorithm vs. programming method, plotted per sorting algorithm. Figure 6.15(a) to (c) shows total number of slices used vs. sorting algorithm, plotted per programming method. The total slice usage as a percentage of the total number of available slices on the Xilinx Kintex-7 160T FPGA is also shown on the right-hand y-axis in Figure 6.15. Both figures use log-log scales.

The slice usage results presented in this section are not directly applicable to FPGAs from other manufacturers since different manufacturers use alternative configurations of lookup tables and flip-flops to construct the logic blocks on their devices. The results are however indicative of which sorting algorithms and programming methods consume more or fewer FPGA logic resources and can still be used to guide the sorting algorithm selection process. Furthermore, where the slice usage approaches the total number of slices on the Kintex-7 160T, the usage results reported may be less representative due to the mapping tool working harder to optimise logic to fit on the device. The sorting algorithms programmed in LabVIEW were implemented and optimised without assistance from National Instruments, using only openly available knowledgebase articles and whitepapers.

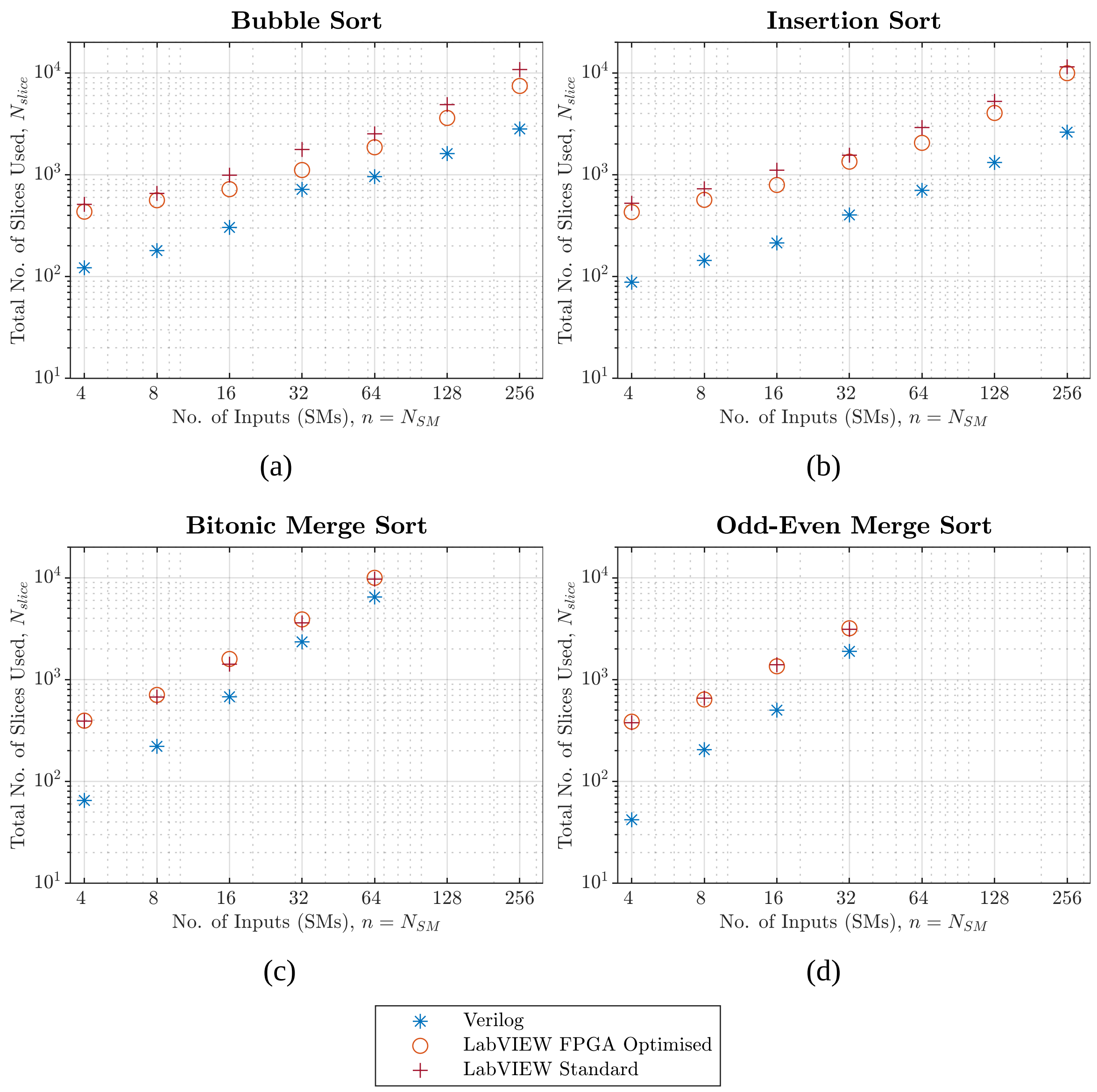


Figure 6.14: Plot of total number of slices used vs. number of inputs vs. programming method for (a) bubble sort, (b) insertion sort, (c) bitonic merge sort, and (d) odd-even merge sort. Note: $\log_2$ x-axis and $\log_{10}$ y-axis scales.

Comparison Based Upon Sorting Algorithm

In the first instance, inspecting Figure 6.14(a) and (b) for bubble sort and insertion sort respectively, it can be seen that the total slice usage for both algorithms grows linearly across all programming methods. This matches the theoretical $O(n)$ (linear) growth rate for the space complexity of bubble sort and insertion sort.

For both algorithms, the increase in slice usage is caused by the additional FPGA memory resources (flip-flops) required to store the input and output arrays as n is increased. Each array element requires the same number of FPGA resources to implement, so increasing the number of array elements leads to linear growth of the total slice usage. Unlike sorting networks (bitonic merge sort and odd-even merge sort), the algorithm structure stays the same and only the number of loop iterations changes as n is increased.

For both bubble sort and insertion sort, the Verilog implementation consistently requires fewer FPGA slices to implement when compared to both LabVIEW implementations. The slice usage overhead for both LabVIEW implementations is due to the additional logic inserted by the LabVIEW FPGA Compilation Tool. This logic is required to ensure that the FPGA design executes according to the LabVIEW dataflow programming paradigm [93]. An approximate value for the difference in slices required per input to the sorting algorithm, $N_{slice-diff}$, between the Verilog and LabVIEW FPGA Optimised implementations can be calculated according to Equation 6.3:

$$N_{slice-diff} = \frac{\Delta N_{slice}}{n} = \frac{N_{slice-LVFPGA} - N_{slice-Verilog}}{n} \qquad 6.3$$

For both bubble sort and insertion sort, $N_{slice-diff}$ fell within the range of +12 to +86 slices per array element, with a mean increase in slice usage of +35 slices. The difference in the number of slices decreased as n was increased and is possibly due to the FPGA synthesis tool working harder to optimise slice usage as the slice usage as a percentage of available slices on the Kintex-7 160T FPGA increased.

Similarly, the LabVIEW Standard implementation consistently requires more FPGA slices than the LabVIEW FPGA Optimised implementation across the range of values of n. This is also due to logic added by the compilation tool which is required to support the higher abstraction level of LabVIEW Standard. Across both bubble sort and insertion sort, $N_{slice-diff}$ between optimised and standard implementations was within the range of +6 to +24 slices with a mean increase in slice usage of +14 slices.

Comparing the two sorting networks in Figure 6.14(c) and (d), both LabVIEW FPGA Optimised and Standard implementations have near identical slice usage across the range of values of n tested. Using Equation 6.3 but comparing the optimised and standard implementations, $N_{slice-diff}$ was in the range −11 to +3 slices with a mean slice difference of −3 slices.

The near identical slice usage of LabVIEW FPGA Optimised and LabVIEW Standard implementations of the sorting networks can be attributed to two factors. Firstly, the same sorting network structure is used for both programming methods, with the same number and sequence of compare-swap operators. Secondly, both implementations require sequencing logic to ensure that downstream compare-swap operators do not execute until the preceding operator has supplied a valid output signal.

In the LabVIEW Standard implementation, the sequencing logic is automatically inserted by the compilation tool to preserve the dataflow execution order of LabVIEW, as outlined previously. In the LabVIEW FPGA Optimised implementation, the sequencing logic was manually programmed in the form of a two-state FSM contained within each compare-swap operator. The manually implemented sequencing logic operates in a similar way to the enable register chain automatically inserted by the compilation tool for the LabVIEW Standard implementation [93] and as a result, requires a similar number of logic resources to implement on the FPGA.

Comparing the Verilog and LabVIEW FPGA implementations in Figure 6.14(c) and (d), again it can be seen that the Verilog implementation consistently has a lower slice usage across all values of n and grows at a slower rate. Across both bitonic merge sort and odd-even merge sort, $N_{slice-diff}$ was calculated in the range +40 to +86 slices, with a mean increase in slice usage of +60 slices. As with insertion sort and bubble sort, the slice difference per array element decreased as n was increased, possibly due to better optimisation of the design by the FPGA synthesis tool as percentage slice usage increases.

Comparison Based Upon Programming Method

As stated previously, the plots presented in Figure 6.15(a) to (c) show the total number of slices used vs. sorting algorithm, plotted per programming method. The right-hand y-axis shows the total number of slices used as a percentage of the total number of slices available on the Xilinx Kintex-7 160T FPGA used in the NI PXIe-7857R.

Comparing the total slice usage of bitonic merge sort and odd-even merge sort with bubble sort and insertion sort, the two sorting networks provide a small reduction in slice usage at n = 4 across all three programming methods. The reduction in slice usage ranges between 23 to 148 fewer slices than bubble sort or insertion sort, representing 0.09 % to 0.58 % of total available slices on the Kintex-7 160T FPGA. This advantage is lost at values of $n \geq 8$, beyond which both sorting networks consume more slices than either bubble sort or insertion sort.

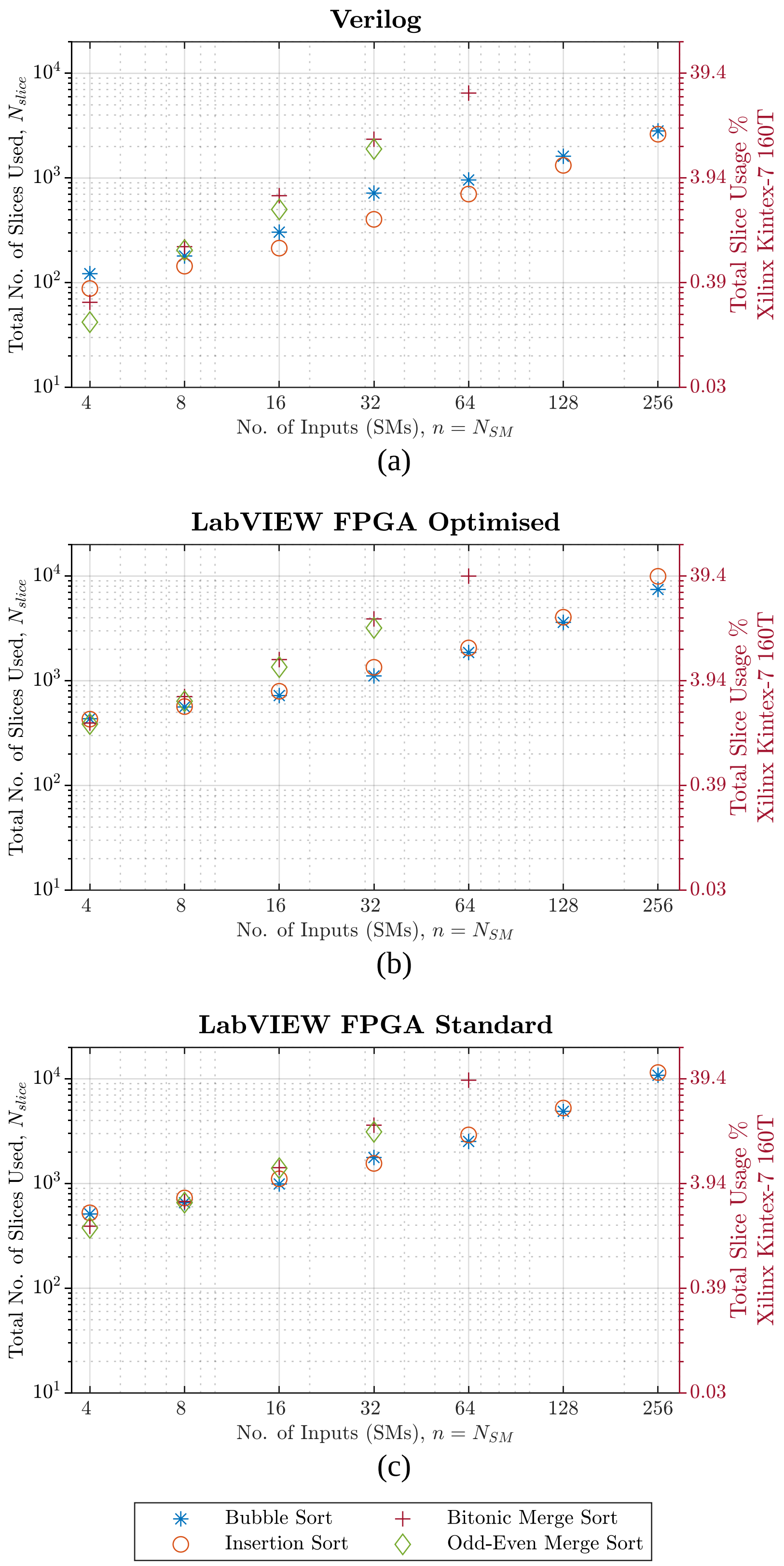


Figure 6.15: Plot of total number of slices used vs. number of inputs vs. sorting algorithm for (a) Verilog, (b) LabVIEW FPGA Optimised, and (c) LabVIEW Standard. The right-hand *y*-axis corresponds to total slice usage as a percentage of available slices on the Xilinx Kintex-7 160T FPGA. Note: $\log_2$ *x*-axis and $\log_{10}$ *y*-axis scales.

The high growth rate of slice usage for both sorting networks is due to the sorting network structure changing as n is increased and more compare-swap operator blocks are added to the sorting network. The number of compare-swap operators required grows according to $O(n \log^2 n)$, as stated in Section 4.1.3, which is greater than the $O(n)$ (linear) space complexity of bubble sort and insertion sort. The higher slice usage growth rate for both sorting networks results in between 25 % to 39 % of total available slices being consumed for bitonic merge sort at n = 64 across all three programming methods. A similar percentage slice usage is reached only at n = 256 for bubble sort and insertion sort when programmed in LabVIEW FPGA Optimised or LabVIEW Standard.

As can be seen in Figure 6.15, odd-even merge sort has a consistently lower slice usage when compared to bitonic merge sort. Across all three programming methods and all values of n, $N_{slice-diff}$ between bitonic merge sort and odd-even merge sort is in the range of −2 to −22 slices with a mean reduction in slice usage of −9 slices. This reduction in slice usage is due to odd-even merge sort requiring fewer compare-swap operators per stage of the sorting network. Coupled with having the same or lower execution time than bitonic merge sort as shown in Figure 6.1, odd-even merge sort is preferable to bitonic merge sort where a sorting network is used for the CBC sorting algorithm

The high slice usage of sorting networks means that there may be insufficient space to implement the CBC loop for even a single converter arm on an FPGA such as the Kintex-7 160T, dependent upon the number of inputs required (recalling that $n = N_{SM}$). Several authors have explored factorising sorting networks by reusing compare-swap operators to reduce logic resource usage, at the expense of increased execution time [66, 67, 71]. Alternatively, an FPGA with more logic resources may be required. Furthermore, manually implementing sorting networks for values of $n \geq 64$ is increasingly difficult and error prone. As a result, it may be necessary to explore the use of automated sorting network generation tools [89, 94] to implement sorting networks for larger values of n.

When compared with each other using the same programming method, bubble sort and insertion sort require a similar total number of slices at each value of n. Across all three programming methods, $N_{slice-diff}$ is in the range −9 to +10 slices, which is 0.039 % of total slices available on the Kintex-7 160T FPGA. Since slice usage is similar between both algorithms, the choice between whether to use bubble sort or insertion sort for CBC should then instead be made based upon which algorithm has the desired execution time.

Despite the lower slice usage growth rate of bubble sort and insertion sort, both sorting algorithms approach between 29 % to 45 % slice usage on the Kintex-7 160T FPGA at n = 256 when programmed in LabVIEW. As a result, it may still only be possible to fit CBC loops for two converter arms on a single Kintex-7 160T FPGA at n = 256, allowing space for other control functions in addition to the CBC loop itself. Since the growth in slice usage for both algorithms is due to the increase in memory elements required to store the input and output arrays, it is impossible to factorise the algorithms to reduce slice usage. An FPGA with more logic resources may be required dependent upon the application specifics, such as: N_{SM}, the number of arms to be controlled by a single FPGA, and the need to implement other control functions on the same FPGA.

Finally, implementing any of the sorting algorithms in Verilog leads to a significant reduction in slice usage across all values of n, when compared to either LabVIEW programming method. Despite this, the implementation difficulty is higher for Verilog, so a trade-off is required between development time and the need to fit all control functions onto the chosen FPGA.

6.6 Summary

The performance of sorting algorithms used in the CBC loop of an MMC have been the subject of limited research to date and have been assumed to be a trivial component in the overall CBC loop. This is despite the fact that sorting is a computationally complex operation, which introduces a time delay to the CBC loop, which may also be non-deterministic in the case of classical sorting algorithms. These factors can degrade capacitor balancing performance.

This chapter has provided a comprehensive evaluation of the performance of a selection of sorting algorithms and sorting networks for capacitor balancing control. A range of sorting algorithms have been tested, focussing upon algorithms which have been mentioned in the open-access literature. The programming methods and control hardware targets which were used to test the sorting algorithms have been chosen to be industrially-representative to ensure the results are applicable to MMCs with different ratings.

Three CBC methods which require a fully sorted list of SM capacitor voltages have been used to generate synthetic capacitor voltage data for three converter operating scenarios. The sorting algorithm execution time and CBC loop sampling period results show the inter-dependence of the chosen sorting algorithm, CBC method, and converter operating scenario. When designing the CBC loop, careful consideration must be given to these factors to ensure that SM capacitor voltages remain balanced and within tolerance.

7 Converter Hardware Prototype

This chapter provides an overview of the reduced-scale converter hardware prototype (CHP) used as the experimental platform for hardware validation during the project. The CHP was constructed as part of a previous PhD project in the Department as documented in [25] and [95]. Extensive changes have been made to the CHP control software during this project to facilitate the research objectives; these are explained in more detail in Chapter 8.

In this chapter the CHP original design objectives are outlined first, followed by the electrical components and ratings, with reference to [25]. Whilst modification of the CHP electrical hardware was not generally required, one significant modification to resolve an issue identified in [25] was made during this project; this is described for documentation purposes. A high-level overview of the distributed control architecture is then provided using a top-down approach, and the individual pieces of control hardware and the communication links between them are outlined.

7.1 Design Objectives

Prototype hardware MMCs constructed in academia or in industry are typically designed for research in a specific area such as: power system and multi-terminal networks, novel converter and SM topologies, fault and protection methods, or converter control architectures [82]. Designing for a specific objective allows for in-depth studies in the chosen area, as well as reducing complexity and cost by simplifying the design in areas which are not the main focus.

This CHP has been designed for research into the effect of industrially representative control and communication implementations upon internal converter dynamics. This includes the effects of non-ideal behaviour such as computation and communication delays, jitter, and packet loss which are present in real-world digital control systems. To enable this type of research, the CHP uses a distributed control architecture typical of that found in an industrial-scale HVDC MMC [82].

The electrical configuration and control architecture of the CHP have been designed to be reconfigurable. This enables research into a range of converter topologies, SM configurations and control implementations.

7.2 Hardware Overview

Constraining the research objective of the CHP to control and communication studies allowed for a system with lower electrical ratings than prototype converters targeted at other study types. Despite having significantly lower ratings, the CHP has been designed so that converter- and SM-level dynamics reflect those of an HVDC MMC. Furthermore, the firing signals sent to the SM switch gate drivers are the same regardless of the chosen power level, therefore the scalability of this research to MMCs operating at higher power levels is not adversely affected.

During the CHP design process, the DC bus voltage, V_{dc}, was chosen first and the ratings for the other system components selected around it [25]. A B&K Precision PVS60085MR [96] programmable DC supply is used as the DC bus; this has a maximum output voltage of 600 V and a maximum output current of 8.5 A. The budget for the CHP construction project allowed for building a system with 48 SMs total. Initially assuming a single-phase converter with 24 SMs per arm, the nominal submodule voltage, V_{SM-nom}, was calculated according to Equation 7.1.

$$V_{SM-nom} = \frac{V_{dc-max}}{N} = \frac{600}{24} = 25\text{ V} \tag{7.1}$$

By limiting the DC bus voltage to 200 V, the CHP can be configured as a three-phase, 9-level MMC with 8 SMs per arm. This configuration has been used throughout this research and is reflected in the nominal electrical ratings shown in Table 7.1. Since the behaviour of the AC side system is not a focus of this work, the AC side load is a simple star-point resistive load. The DC neutral point is a virtual neutral point formed by a potential divider across the DC bus.

Parameter	Value	Units
Nominal DC bus voltage, V_{dc-nom}	200	V
Number of phases	3	
Number of submodules per arm, N_{SM}	8	
Submodule voltage, V_{SM-nom}	25	V
Apparent power, S_{nom}	1	kVA
Arm inductance, L_{arm}	10	mH
AC output voltage (phase-neutral, RMS), V_{ac}^{RMS}	71	V
AC output current (per-phase, RMS), I_{ac}^{RMS}	4.7	A
AC side load (per-phase), R_{load}	200	Ω

Table 7.1: CHP nominal electrical ratings (present configuration)

The CHP submodules, National Instruments (NI) PXIe control chassis, DC supply and all ancillary equipment are housed in a 19-inch rack cabinet as shown in Figure 7.1. This ensures that points at a high potential are protected from accidental contact during operation and reduces the likelihood of faults caused by movement of components. The AC load is mounted on the side of the rack for cooling purposes. A PC-based human-machine interface (HMI) outside the cabinet is used for supervisory control of the CHP.

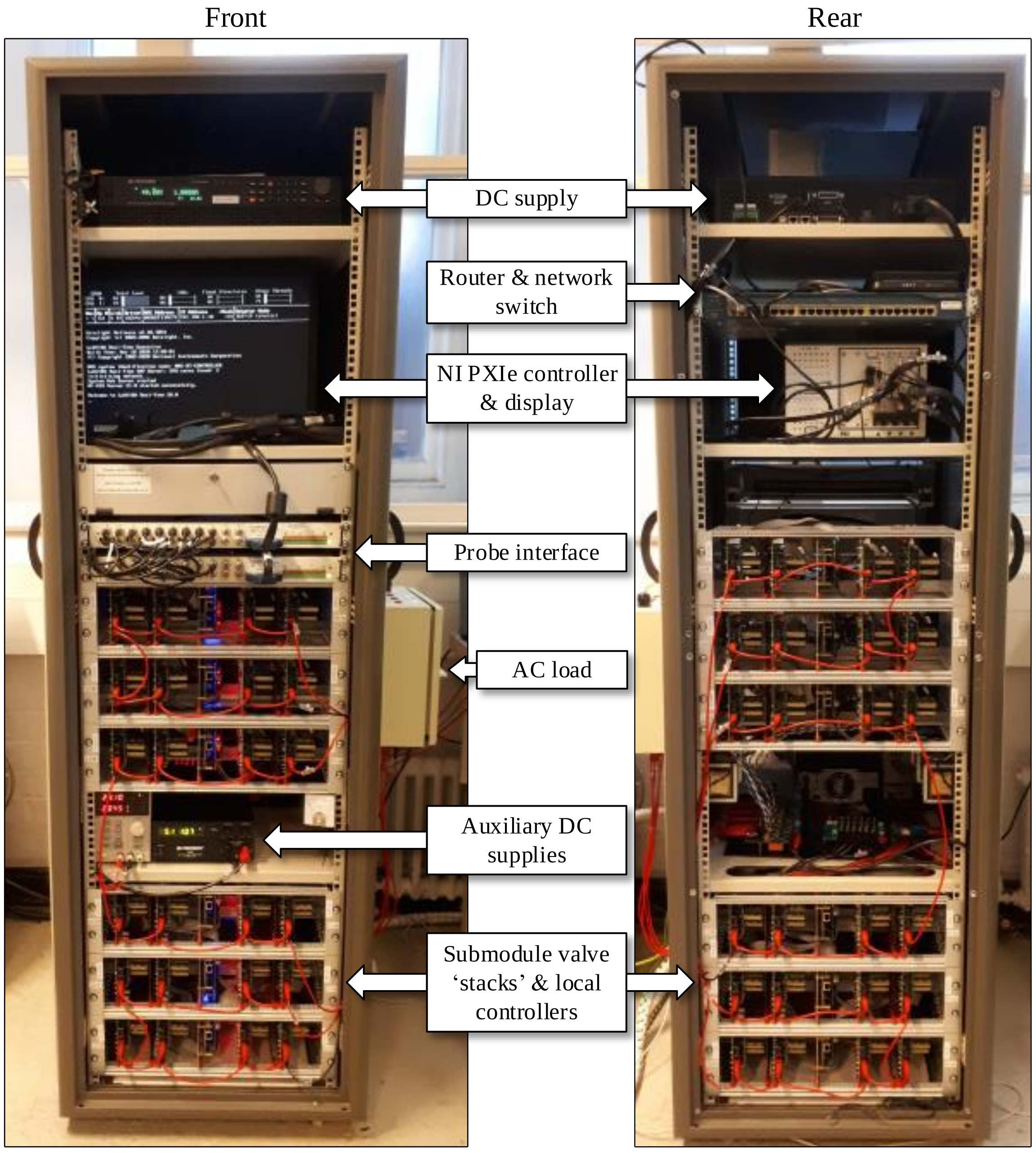


Figure 7.1: Annotated CHP system hardware layout, front (left) and rear (right)

In addition to the main system components shown in Figure 7.1, the CHP rack contains the communication network hardware for the distributed control architecture and other converter support circuitry. These components are listed in Table 7.1 along with their functions and will be referred to in subsequent sections.

Component	Description/Function
Auxiliary DC supplies and power distribution PCBs (×2)	• 5 V: SM local controllers (LC), logic-level circuits on SMs, fibre optic breakout board, power interface board • 24 V: SM gate drive circuits
Power interface board (PIB)	• Connection point for DC supply, 6× arms, AC load • DC and AC side breaker relays • Charging resistors and bypass relays • Hall-effect current sensors for I_{ac}, I_{dc}, and I_{arm}
NI BNC2090A probe interface (×2)	• Data acquisition (DAQ) card input-output (IO) breakout into BNC connectors for I_{arm} current clamp probes and $V_{ext(abc)}$ external reference input
Arm current clamp probes (×6)	• Pico Technology TA189 [97]
Fibre optic breakout board (FOBB)	• Fibre optic transmitter/receiver and driver circuits for SM LC communication interface • Connector for arm control unit FPGA
Router and network switch	• Wired Ethernet network for NI PXIe controller, main DC supply and HMI PC

Table 7.2: List of CHP auxiliary hardware and functions

7.2.1 Submodule Overview

The SMs in the CHP were designed around the ratings listed previously in Table 7.1 with an additional safety margin to minimise the risk of damage to SMs due to incorrect connections or control configuration during experimental work. All 48 SMs are identical and can be individually configured for half- or full-bridge operation via a physical jumper cable. The HB/FB topology was originally chosen since HB/FB-SMs are widely used in industry and the ability to reconfigure SMs allows for research into hybrid HB/FB MMC topologies if desired [25]. All SMs were operated in HB mode throughout this project since converter topologies were not the research focus. The electrical ratings of the CHP SMs are summarised in Table 7.3.

Parameter	Nominal	Maximum	Units
Submodule voltage, V_{SM}	25	57	V
Submodule capacitance, C_{SM}	3000	–	µF
Submodule current, I_{SM}	5	8	A

Table 7.3: CHP submodule electrical ratings

A simplified schematic of a CHP SM is shown in Figure 7.2, along with the key specifications of selected SM components in Table 7.4. The SM output voltage is taken from banana jack connections A-B in HB-SM mode or across connections A-C in FB-SM mode. Metal-oxide-semiconductor field-effect transistor (MOSFET) switches are used instead of insulated-gate bipolar transistors (IGBT) as used in industrial-scale SMs due to having a lower forward voltage drop in this application than IGBTs. This ensures that SM conduction loss and switching behaviour remains representative when operating at the reduced voltage level of the CHP.

The required SM capacitance was calculated to ensure that the capacitor voltage ripple error is below 10 % when the CHP is operating at or below its peak apparent power rating of 1.8 kVA [25]. This yielded a maximum value of C_{SM} = 2480 µF, which was rounded to 3000 µF and is implemented using three 1000 µF capacitors connected in parallel to increase the ripple current rating.

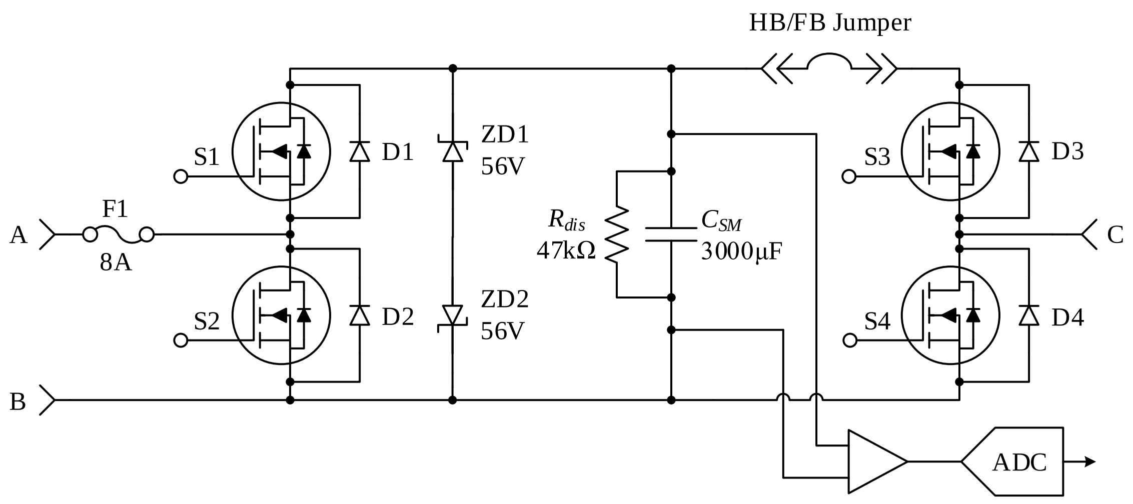


Figure 7.2: CHP submodule simplified schematic

Component	Key Specifications
Main switches, S1-S4	N-channel MOSFET V_{DS} = 60 V, I_{DS} = 75 A, $R_{DS(on)}$ = 6 mΩ
Submodule capacitor voltage ADC	Maximum sample rate 1 mega samples per second (MSa/s), 10-bit resolution
Over-voltage protection	2× 56 V back-to-back Zener diodes
Over-current protection	8 A fast-blow fuse
Discharge resistor, R_{dis}	47 kΩ

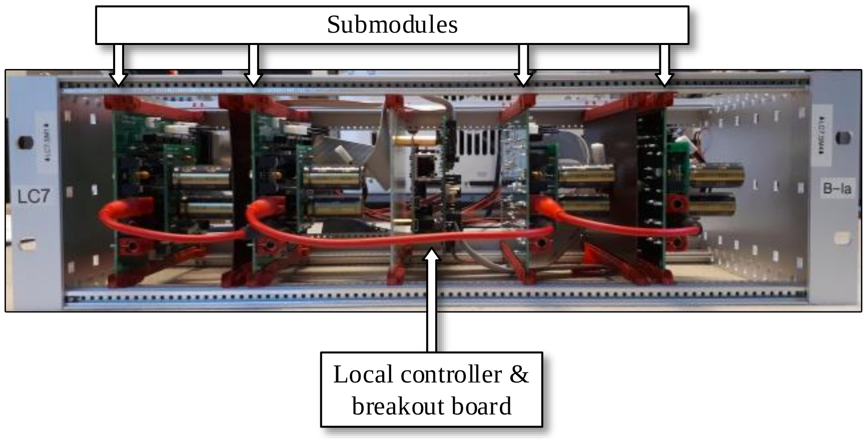
Table 7.4: CHP submodule component key specifications

Over-current protection of the SM is provided by an 8 A fast-blow fuse, whilst bi-directional over-voltage protection is provided by two back-to-back Zener diodes which will clamp the SM voltage at 56 V. A 47 kΩ resistor connected across C_{SM} ensures that the SM is discharged when power is disconnected. Additional support circuitry is also installed on each SM to provide:

- Isolated DC-DC power supplies for SM switch gate drivers and the SM capacitor voltage analogue-to-digital (ADC) converter
- Isolated MOSFET gate drive signals
- Opto-isolators for the ADC to LC serial peripheral interface (SPI)
- SM capacitor voltage signal conditioning (downscaling and low-pass filtering) for ADC input

The support circuitry on the SMs is powered from external 5 V and 24 V power supplies, rather than from energy harvested from SM capacitor as is the case in industrial-scale SMs. This decision was made due to the relatively high power consumption of the SM auxiliary circuits (approximately 2 W per SM) relative to the stored SM energy (less than 1 J), which would degrade CBC performance and power control response [25]. Furthermore, energy harvesting from C_{SM} would complicate the CHP start up and pre-charging routines.

The CHP SMs are sized according to standard Eurocard printed circuit board (PCB) dimensions to facilitate easy mounting. Four SMs and one LC are mounted in a 19-inch rack Eurocard chassis and form a CHP ‘valve’ stack, as shown in Figure 7.3. SM control signals connect to the LC via multi-way board-to-board ribbon cables and a breakout board on top of the LC field-programmable gate array board.



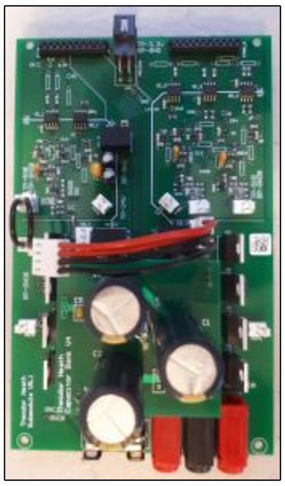


Figure 7.3: Four SM valve ‘stack’ and local controller (left) and individual SM (right)

7.2.2 Hardware Modifications

The overall hardware configuration of the CHP was maintained in its original state for the duration of this project, however some modifications were made to resolve issues identified in [25], notably, noisy and inaccurate current measurements from the Hall-effect sensors on the PIB.

The Hall-effect sensors on the PIB are used to measure I_{ac}, I_{dc}, and $I_{u,l}$ in the CHP and output a voltage proportional to the current flowing through the sensor. In the present configuration only the arm current measurements are required; these are used by the CBC loop. The AC output and DC supply current measurements are not used in converter control loops but may be logged for analysis.

Further troubleshooting during this project showed a DC offset on the sensor outputs which was not present in the arm current when measured with an external clamp type current probe. This offset changed dependent upon the AC and DC breaker relay states and could not be nulled due to its varying nature. A more detailed description of the source of the issues and steps taken to resolve them can be found in Appendix E.

The DC offset in the arm current measurements led to a subset of SMs in each arm not balancing correctly, since the CBC loop uses the arm current direction to determine which SMs to insert or bypass. Poor performance of the CBC loop was observed from the measured capacitor voltages when the CHP control software was re-implemented as described in subsequent sections in this chapter.

To provide accurate arm current measurements, six Pico Technology TA189 [97] clamp type current probes were installed in the CHP to replace the measurements from the sensors on the PIB. The probes have a range of 0 to 30 A, a sensitivity of 100 mV/A and were chosen since they are sensitive enough to measure the arm currents in the CHP which are typically less than 500 mA during steady state operation. Two NI BNC-2090A [98] rack mount connector panels were installed in the CHP to provide easy connection of the BNC outputs of the current probes to the DAQ card analogue inputs.

7.3 Control Architecture Overview

As stated in the design objectives at the beginning of this chapter, the CHP uses a distributed control architecture which is representative of the control architectures adopted in many industrial-scale MMCs. In this type of control architecture, converter control functions and IO are split across several hardware targets which are connected using communication links. This differs from a centralised control architecture, where all control functions and IO are implemented on a single central controller. In an industrial setting, a distributed control architecture is necessary as the computational requirements for control and the quantity of IO for SM communication exceeds that of a single centralised controller [95]. Other advantages of a distributed architecture include [25, 99]:

- Increased reliability: no single point of failure and redundancy is easier to implement since processing and communication are distributed and/or modularised.
- Reduction in computational requirements for individual control hardware platforms: this enables use of less powerful (and often lower cost) control hardware, whilst allowing more functions to be implemented using the available processing resources.
- Reduction in IO requirements for individual control hardware targets: IO and DAQ are split across hardware targets.

Implementing a distributed control architecture on a prototype or industrial-scale MMC brings several challenges when compared to a centralised architecture. These include [82]:

- Inter-controller communication interfaces must be defined, implemented, and tested, in addition to converter control loops.
- Special attention must be paid to synchronisation and phase alignment of control loops and data acquisition across control hardware targets.

- Communication interfaces between controllers and/or SMs introduce non-linearities such as variable delays and packet loss.

A high-level block diagram overview of the CHP control architecture is shown in Figure 7.4, with the HMI PC at the top of the diagram and SMs at the bottom. The different control hardware targets used in the CHP are shown, along with the communication links between them and the data passed between hardware targets. Also shown is the programming language used for each hardware target where appropriate.

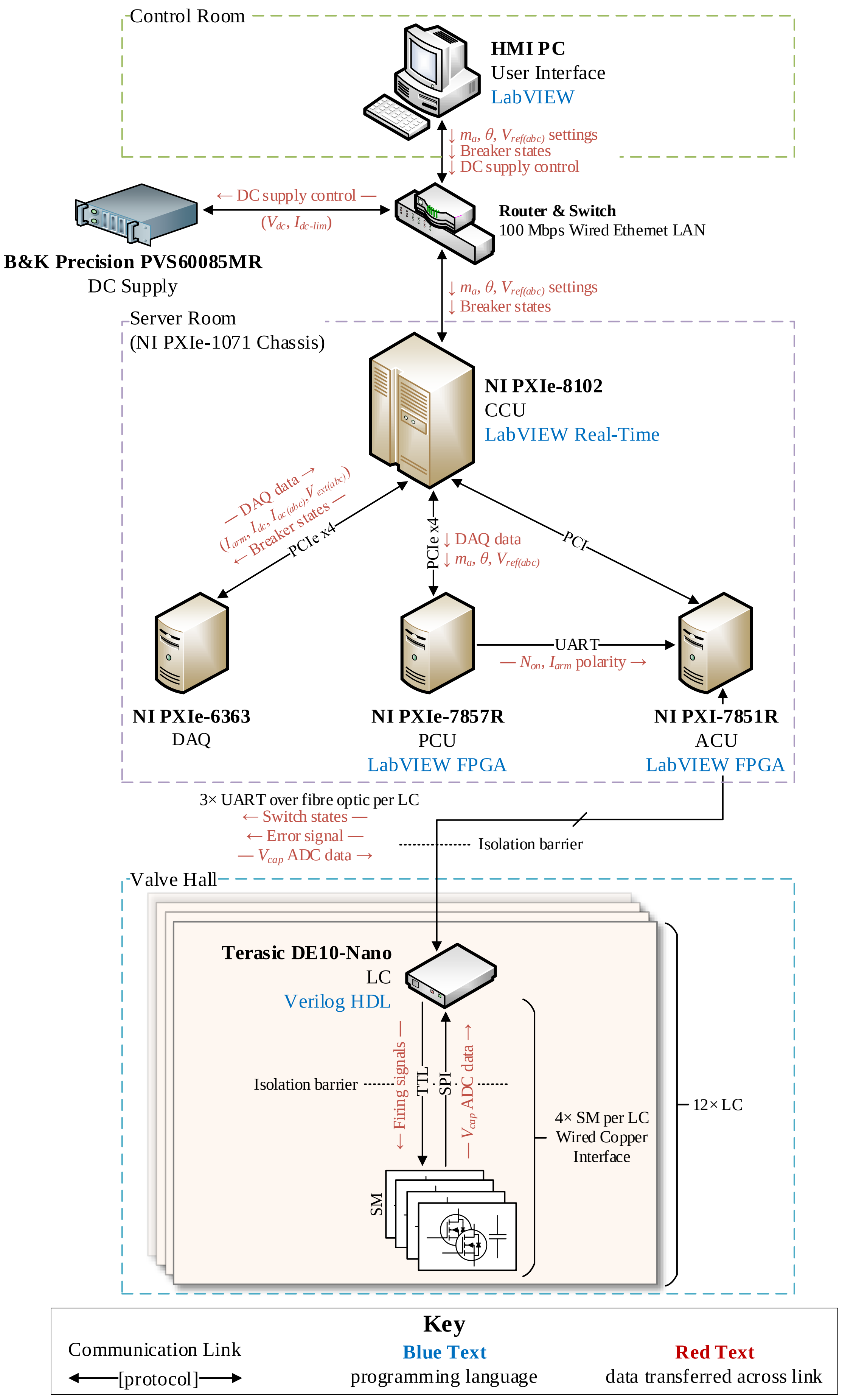


Figure 7.4: CHP distributed control architecture overview

7.3.1 Human-Machine Interface (HMI)

The HMI is hosted on a standard desktop PC and provides a front panel graphical user interface (GUI) for supervisory control of the CHP. The HMI is programmed in LabVIEW and performs the following functions:

- Coordinates converter start up and shutdown.
- Run-time setting of voltage reference modulation index (m_a) and phase (θ_{abc}).
- DC supply control to set output voltage, current and protection limits.

The functions performed by the CHP HMI are broadly equivalent to the control room and station dispatch control level in an industrial-scale MMC. The HMI communicates with other control hardware targets in the CHP over a wired Ethernet local area network (LAN); this is representative of an industrial control architecture where a protocol such as IEC 61850 [100] running over a LAN is typically used for substation control messages [101].

7.3.2 Local Area Network (LAN)

A 100 megabit per second (Mbps) wired Ethernet LAN is formed by a network router and 24-port switch as shown in Figure 7.4 and provides communication between the HMI PC, DC supply and converter control unit.

7.3.3 Converter Control Unit (CCU)

An NI PXIe-8102 embedded controller [83] running the Phar Lap ETS real-time operating system (RTOS) is used as the CCU and is programmed in LabVIEW Real-Time (RT) which is a subset of standard LabVIEW. The CCU handles several functions as follows:

- Control of the phase control unit (PCU) and arm control unit (ACU) FPGAs.
- Configures and starts/stops DAQ card IO tasks.
- Implements data transfer between DAQ card and PCU for analogue measurements.
- Handles data transfer between ACU, CCU and HMI PC for logging.
- Handles control settings transfer from HMI PC to PCU and ACU.

The CCU communicates with the other cards installed in the NI PXIe-1071 chassis [102] over internal peripheral component interconnect (PCI) and PCI Express (PCIe) buses; these are described in more detail in Chapter 8. No control loops are implemented on the CCU itself; instead, the CCU is a host for the other cards installed in the chassis. The PXIe system with installed processing cards is representative of the control equipment installed in the server room in an industrial-scale MMC, albeit at a reduced scale and complexity.

7.3.4 Data Acquisition (DAQ)

An NI PXIe-6363 card [103] is used as the DAQ device in the CHP and provides analogue-to-digital conversion of arm current measurements and the external three-phase reference voltage input. The digital outputs on the DAQ card are used to drive the system breakers installed on the PIB.

7.3.5 Phase Control Unit (PCU)

A NI PXIe-7857R FPGA card [84] is used as the PCU and performs phase-level converter control functions in the CHP. These are: circulating current suppression control, arm balancing, nearest level control, and the external reference phase-locked loop (PLL). In addition to these functions, the PCU also contains an internal three-phase reference voltage generator for direct control of the CHP AC output voltage. The PXIe-7857R uses a Xilinx Kintex-7 160T FPGA and is programmed in LabVIEW FPGA, which is a subset of standard LabVIEW. The PCU communicates with the ACU FPGA over a simplex universal asynchronous receiver-transmitter (UART) interface and sends the six arm current polarity measurements and arm insertion indices (N_{on}) for use by the ACU.

7.3.6 Arm Control Unit (ACU)

The ACU is based on a NI PXI-7851R FPGA card [104] and performs arm-level control functions such as: capacitor balancing control and switch state generation using the data provided by the PCU. The ACU interfaces with the twelve LC FPGAs over a fibre optic interface, with each LC allocated one UART transmit/receive pair and an auxiliary error signal. The PXI-7851R card in the ACU uses a Xilinx Virtex-5 LX30 FPGA and is also programmed in LabVIEW FPGA.

A photograph of the PXIe system with the cards described in the previous sections is shown in Figure 7.5.

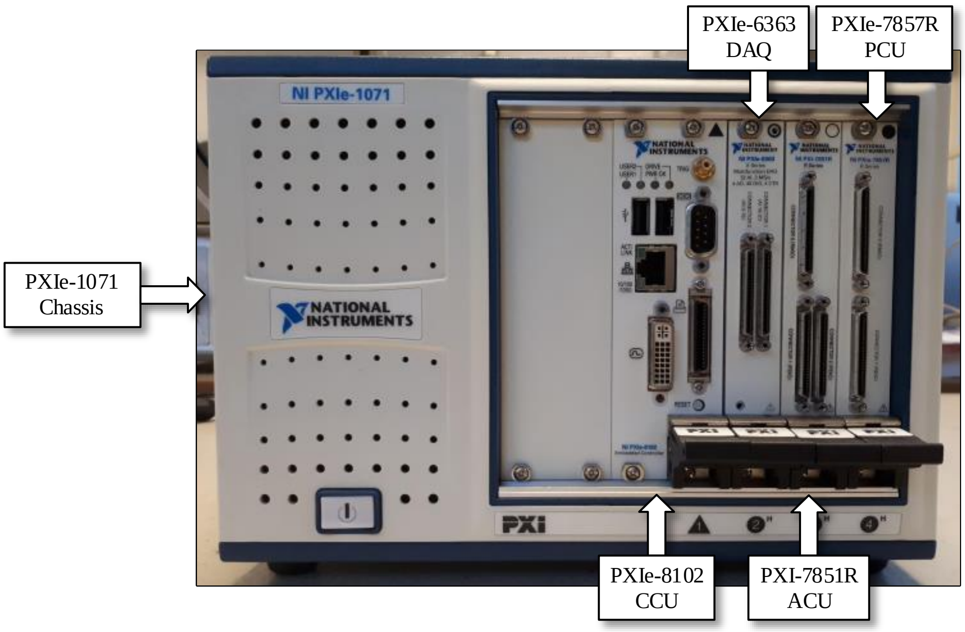


Figure 7.5: NI PXIe system with CCU, DAQ, ACU, and PCU cards (left to right)

7.3.7 Local Controller (LC)

The twelve LCs in the CHP are based on a Terasic DE10-Nano FPGA development board [105] which use an Intel Cyclone V system-on-chip (SoC) and are programmed in the Verilog hardware description language (HDL). The LCs communicate with the ACU over the fibre optic interface and parity check and parse received data packets before sending firing signals to four SMs over a wired interface operating at 5 V logic levels. The LCs also handle SM capacitor voltage acquisition from the SPI ADC installed on each SM; these measurements are transmitted back to the ACU along with any error signals across the fibre optic interface. A photograph of the LC FPGA board and breakout board are shown in Figure 7.6.

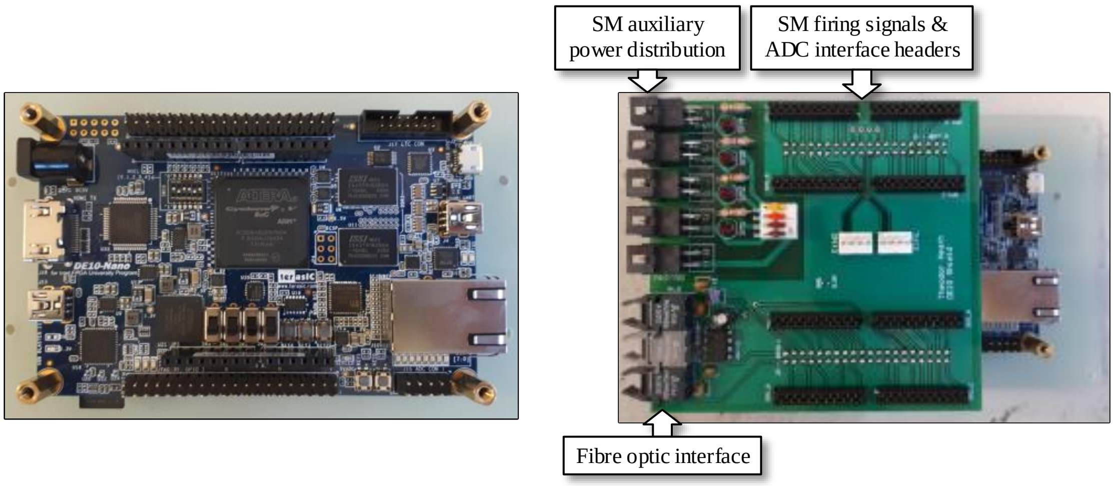


Figure 7.6: Terasic DE-10 Nano Local Controller (left), with breakout board installed (right)

7.4 Summary

This chapter has provided an overview of the CHP electrical configuration and control architecture and serves as a basis for understanding the control system software development and hardware results presented in subsequent chapters. With the exception of replacing the arm current sensors, the electrical configuration of the CHP was left largely unchanged throughout this project due to this not being the research focus. The control architecture overview presented in this chapter is a high-level introduction and will be expanded upon in Chapter 8.

8 Control System Software Development

This chapter provides a detailed description of the development of the converter hardware prototype (CHP) control software undertaken during this project. Whilst the existing CHP control software met the research objectives of the original CHP construction project, it was not able to support the objectives of this research, and as a result, it was decided to rewrite the CHP control software from first principles without reusing any of the existing source code. To deliver the precise timing measurements required by the research objectives of this project, a detailed understanding of the control and timing considerations in the CHP was required; these are described in this chapter.

In this chapter, a brief overview of the LabVIEW programming language is provided first, followed by a review of the existing control software, identifying its limitations. The design objectives of the new control software are then outlined. A detailed description of the NI PXIe system internal architecture is then provided, building on the control architecture overview in Chapter 7. This is used as a foundation for explaining the implementation specifics of the control system software written during this project, which is documented in Section 8.5 and subsequent sections. The most recent version of the CHP control software is available in the online supplementary repository [26].

8.1 Overview of the LabVIEW Programming Language

As shown in Figure 7.4 in Chapter 7, the control software running on the human-machine interface (HMI) PC, converter control unit (CCU), phase control unit (PCU), and arm control unit (ACU) is written in LabVIEW. LabVIEW is a graphical block diagram-based programming language with inherent support for parallel execution developed by National Instruments and was used for the existing CHP control software developed in [25]. It was also used in this project because it is natively supported by the CCU, data acquisition (DAQ), PCU, and ACU hardware targets. Furthermore, the graphical block diagram-based programming approach allows for quicker control software development and debugging when compared to conventional text-based programming languages. Faster development is also facilitated by the wide range of library functions provided by LabVIEW which allow complex control functions and communication protocols to be implemented with relative ease.

In LabVIEW terminology, programs are referred to as virtual instruments (VI) and have two components: a front panel and a block diagram. The VI front panel contains controls (inputs) and indicators (outputs) and is the graphical user interface (GUI) for the VI, whilst the block diagram contains the graphical 'code' which defines the processing operations performed by the VI. To allow for code re-use and division of processing functions into defined blocks, VIs can be split into 'subVIs' which are called by a parent VI – this concept is analogous to functions or subroutines in a conventional text-based programming language.

Most programming constructs in LabVIEW are platform-independent – that is, the same VI will usually execute on different hardware targets without requiring modification. Despite this, several variants of LabVIEW are available which provide additional constructs to exploit the underlying processing architecture of the hardware target (eg. x86 CPU, RTOS, FPGA), or remove functions which cannot be implemented on a particular target. The LabVIEW variants used on the hardware targets in the CHP are listed in Table 8.1.

Hardware Target	**Underlying Architecture (OS)**	**LabVIEW Variant**
HMI (Desktop PC)	Intel x86 CPU (Windows 10)	Standard
CCU (PXIe-8102)	Intel x86 CPU (Phar Lap ETS RTOS)	Real-Time
PCU (PXIe-7857R)	Xilinx Kintex-7 FPGA	FPGA
ACU (PXI-7851R)	Xilinx Virtex-5 FPGA	FPGA

Table 8.1: Summary of CHP control hardware targets and LabVIEW variants

Using LabVIEW provides several advantages when writing control system software, as outlined previously. Whilst these may accelerate initial development, a detailed understanding of LabVIEW and hardware target-specific constructs is still required to implement complex control and communication functions such as those in the CHP. Moreover, optimising code for speed or resource usage requires knowledge of the underlying architecture of the control hardware target, in addition to LabVIEW itself.

8.2 Review of Existing CHP Control Software

The original control software for the CHP was developed alongside the control structure and hardware construction process as documented in [25]. Whilst the existing control software functioned as desired, several limitations and code bugs were present and were identified in [25]. These issues were not addressed in the original CHP construction project since control software development was not the main focus; instead, only the minimum required functionality to obtain the necessary measurements was implemented. A code review was carried out during this project which identified further limitations to the existing control software; these are summarised as follows:

- Limited (< 4 %) remaining FPGA logic resource on PCU and ACU FPGAs:
  The existing implementation in LabVIEW FPGA consumed a high percentage of available resources (flip-flops and lookup tables) on the PCU and ACU FPGAs. This meant that the additional functionality required for measurements during this project would not fit on the FPGA.
- No synchronisation or phase alignment between data acquisition and control loops on the same or different hardware targets in the PXIe system:
  In the existing software, sample clocks were generated using separate local timebases which were not phase aligned. This led to a large, variable delay (jitter) on the total system delay in the CHP of over 1000 μs [25], caused by asynchrony between control loop output references updates and sampling points.
- No runtime configurability of data acquisition and control loop sampling rates:
  Sampling rates were hard coded into the FPGA VIs and could not be updated prior to converter start-up or during runtime without re-compiling the associated FPGA VI, a process which takes between 10 to 15 minutes, slowing experimental work.
- Limited observability and data logging of CHP internal electrical quantities:
  Whilst internal measurements of arm current and SM capacitor voltages were received and processed by the control software, it was not possible to plot these on the HMI PC or log to a file during runtime for post-run analysis.

8.3 Design Objectives

The design objectives of the CHP LabVIEW control software rewrite are listed below:

- Reduce total logic resource usage on FPGA hardware targets (PCU and ACU) by utilising LabVIEW FPGA-specific constructs and code optimisation techniques.
- Implement synchronisation and phase alignment between data acquisition and control loops executing on different hardware targets (DAQ, ACU, PCU).
- Implement functionality to configure the sampling rate and phase alignment of individual control loops from the HMI PC at runtime without re-compilation of FPGA VIs.
- Implement digital logic-level outputs for precise measurement of control loop execution time using an external logic analyser.
- Implement functionality to plot and log internal measurements in the CHP at runtime.

Based on the code review and the additional functionality required by this project, it was decided to rewrite the CHP LabVIEW control software from first principles without reusing any of the existing source code. This decision was taken due to the design objectives necessitating a fundamentally different approach to control software implementation. The existing control software was however used as a reference throughout the development process. It was not necessary to modify the Verilog HDL code running on the twelve LC FPGAs significantly during this work.

8.4 PXIe System Internal Architecture

As shown in the high-level overview of the CHP control architecture in Figure 7.4, several communication links exist inside the NI PXIe system itself. These links are made available to cards installed in the PXIe chassis via a backplane and can be used by the programmer to provide communication and synchronisation functions between hardware targets.

A more detailed description of the internal architecture of the PXIe system is provided in this section as a basis for understanding how the new control software leverages the underlying architecture to achieve the design objectives. A diagram of the internal architecture in the form used by the existing control software is shown in Figure 8.1 and will be used to describe the features common to both the existing and new control software implementations.

An updated diagram which shows the additional signals and functionality provided by the new software is shown in Figure 8.2, with the differences highlighted in red.

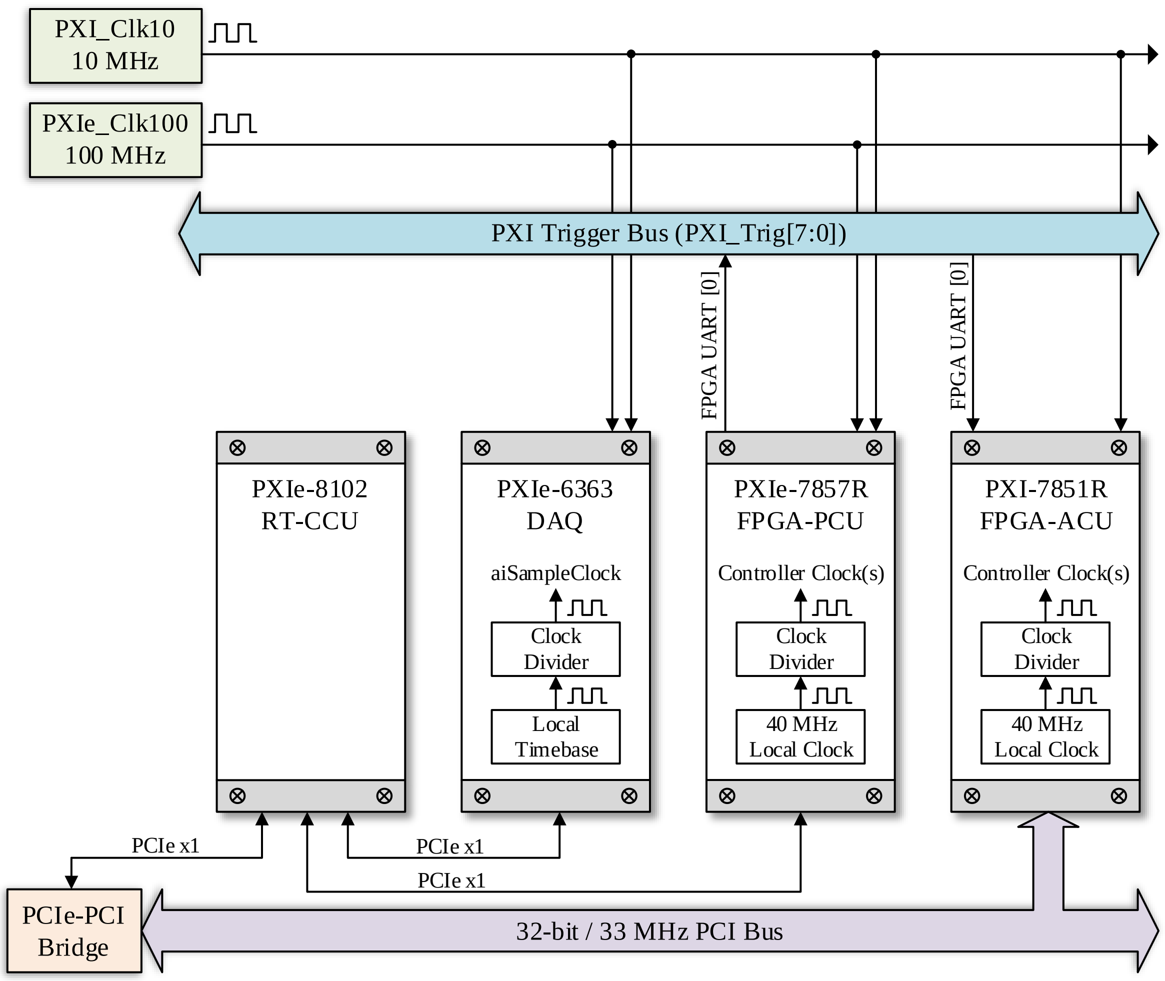


Figure 8.1: CHP PXIe system internal architecture (existing control software)

8.4.1 CCU Host Communication Buses

The primary method of communication between the PXIe-8102 embedded controller (CCU) and the DAQ, PCU, and ACU is via either the PCIe or PCI buses provided on the PXIe system backplane. The CCU acts as the bus host and coordinates data transfer under control of the LabVIEW VI running on the CCU.

The PXI-7851R ACU communicates with the CCU over the PCI bus via a PCIe-PCI bridge as shown in Figure 8.1. PCI is a legacy 32-bit parallel bus standard which operates at 33 MHz and provides a maximum bandwidth of 132 megabytes per second (MB/s) [83].

The PXIe-6363 DAQ and PXIe-7857R PCU communicate with the CCU directly over the PCIe bus as shown in Figure 8.1. Unlike PCI, PCIe is a serial bus standard which is still actively supported and provides a significantly higher maximum bandwidth of 6 gigabytes per second (GB/s) [83]. The PCI and PCIe buses are used by the CCU embedded controller to provide several capabilities as part of the PXIe system:

- Device configuration (eg. PCU/ACU FPGA code download and run/halt, DAQ card IO routing and task configuration).
- Runtime data transfer to/from CCU embedded controller and DAQ, PCU, and ACU. This data may be DAQ samples or control settings from other devices such as the HMI PC.

8.4.2 Clock Sources

Several clock sources are available to cards installed in the PXIe chassis; these can be categorised as local (internal) clock sources or external clock sources. In NI data acquisition terminology, clock sources are also referred to as timebases and the two terms will be used interchangeably here.

The DAQ, PCU, and ACU each have one or more local clock sources which can be used as the primary clock for functions on the hardware target. In the case of the DAQ card, three timebases (100 kHz, 20 MHz, and 100 MHz) are available internally [106] and can be used to generate the ADC sample clock for analogue inputs (*aiSampleClock*) via a programmable clock divider as shown in Figure 8.1. Corresponding internal sample clock signals exist for analogue output and digital IO functions. Unless specified by the programmer, the clock source for DAQ IO operations is chosen automatically by the NI DAQmx driver software.

For the PCU and ACU FPGA cards, a 40 MHz local base clock is provided on each card and by default is used by the LabVIEW FPGA VI global clock for all FPGA logic. The base clock can be multiplied or divided to different frequencies using an onboard PLL. It is only possible to generate clock frequencies in the order of MHz using the PLL, therefore counter-based clock divider or delay functions must instead be used to generate the relatively low sampling frequencies (tens of kHz) required by the control loops in the CHP.

The local clocks available on each hardware target are not phase locked even when configured to operate at the same frequency, since each card uses its own local oscillator. This leads to a variable delay between the AC reference and converter AC output waveform, as observed with the existing CHP control software [25].

To provide clock synchronisation capability across hardware targets, the PXIe system distributes two global clock signals to all slots in the chassis. These are named *PXIe_Clk100* and *PXI_Clk10*, operate at 100 MHz and 10 MHz respectively, and are precisely matched for jitter and skew across the chassis backplane [102]. The clock signals can be routed to internal functional blocks under the control of the LabVIEW VI running on the hardware target; this capability was used in the software rewrite and is described in the following sections.

8.4.3 Other Connections

In addition to the buses and clock signals outlined previously, the PXIe chassis provides 8 signals connected to each card slot via the PXI trigger bus (*PXI_Trig[7:0]*). This is a shared bus which can be used for passing synchronisation and timing signals between cards. Unlike the PCI and PCIe buses, signals and protocols routed onto the PXI trigger bus are entirely user-defined in the LabVIEW VIs running on the hardware targets.

In both the existing and new CHP control software implementations, one PXI trigger line (*PXI_Trig[0]*) is used for a simplex UART communication link from the PCU to the ACU. This is used to transfer the arm current polarity and arm insertion indices (N_{on}) calculated by the PCU to the ACU FPGA. The PCU and ACU FPGAs cannot communicate directly using PCI or PCIe since they are connected to different buses and communication via the CCU would introduce undesirable delay and jitter.

As stated in the introduction to this section, Figure 8.2 shows the PXIe system internal architecture with the additional signals implemented and used by the new control software; this will be referenced in subsequent sections in this chapter.

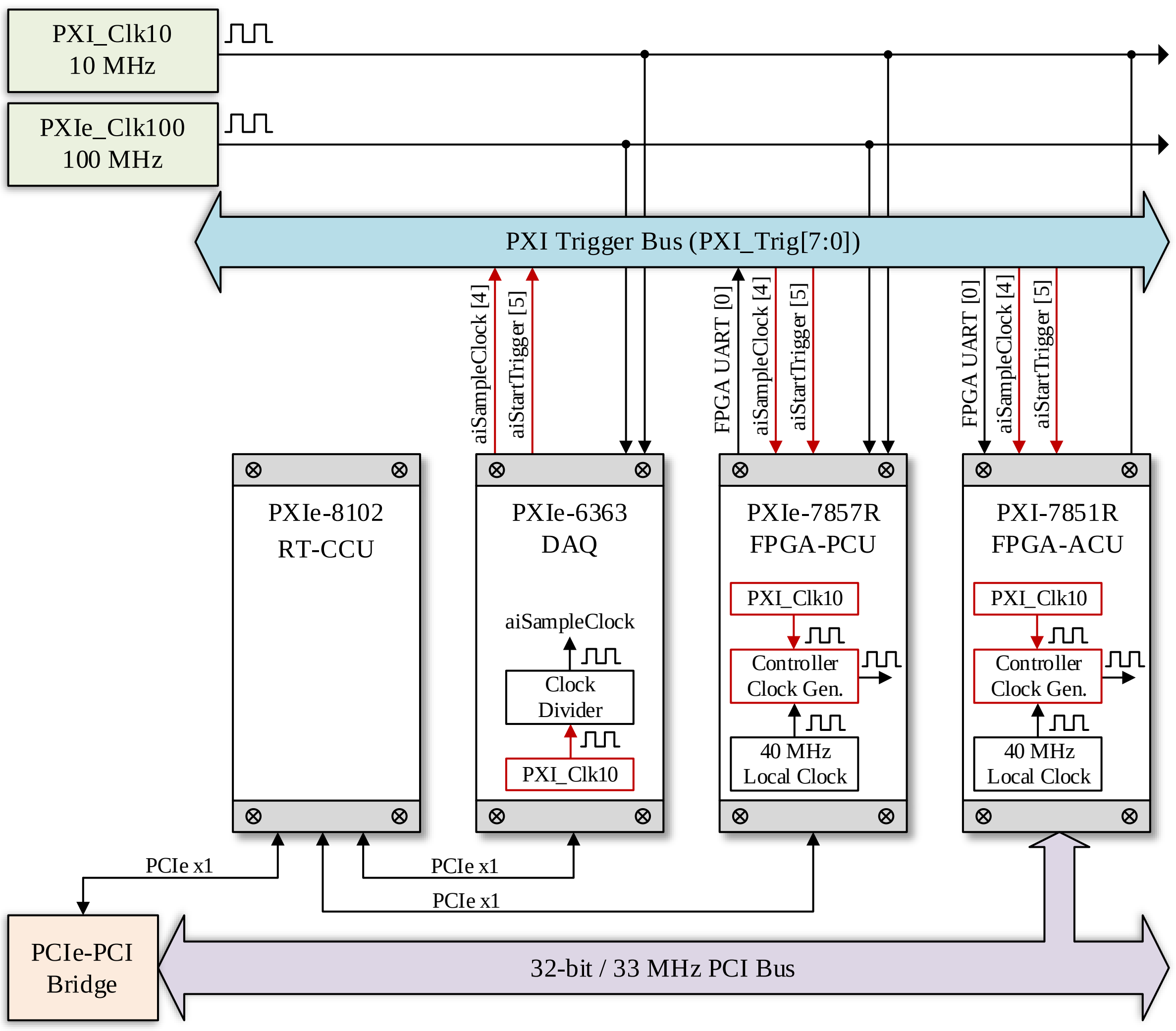


Figure 8.2: CHP PXIe system internal architecture (new control software)

8.5 Controller Sample Clock Generation

To enable research into the effects of control loop sample clock selection and controller execution delay upon total system delay, the LabVIEW FPGA code running on the PCU and ACU was re-written to provide flexible runtime configuration of control loop sample clocks from the HMI PC. The same architecture for the sample clock generator logic is duplicated for each control loop implemented on the PCU and ACU FPGAs; the general architecture is shown in Figure 8.3.

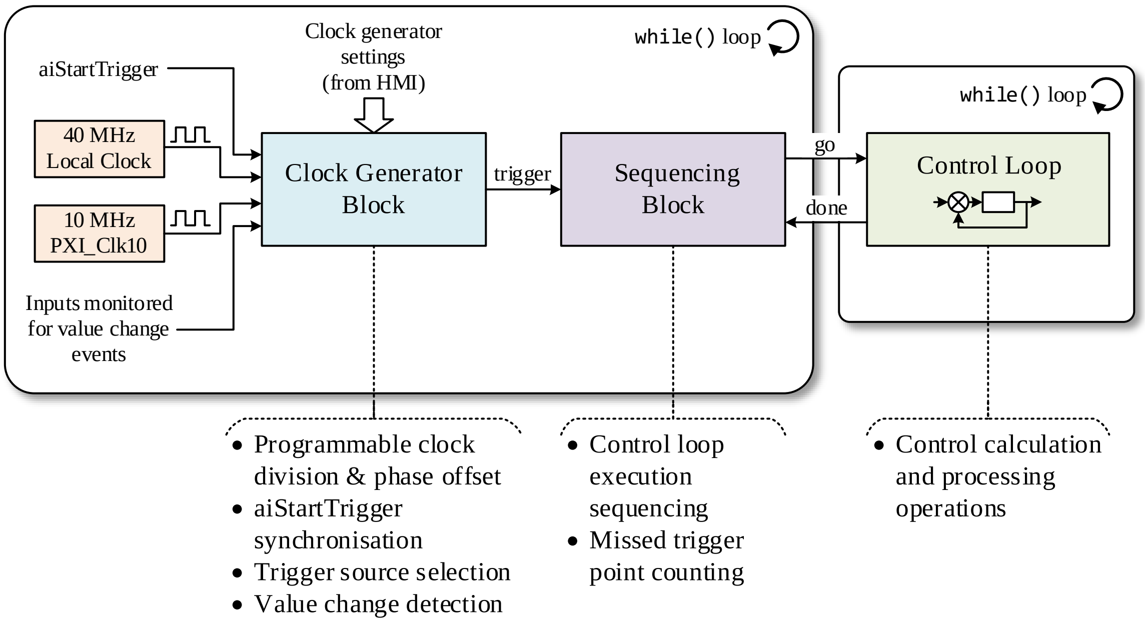

while() loop
Clock generator settings (from HMI)
aiStartTrigger
40 MHz Local Clock
10 MHz PXI_Clk10
Inputs monitored for value change events
Clock Generator Block
trigger
Sequencing Block
go
done
while() loop
Control Loop
• Programmable clock division & phase offset
• aiStartTrigger synchronisation
• Trigger source selection
• Value change detection
• Control loop execution sequencing
• Missed trigger point counting
• Control calculation and processing operations

Figure 8.3: Controller sample clock generator functional overview

As shown in Figure 8.3, sample clock generation and execution sequencing are decoupled from the control loop itself by placing the code in separate, indefinite while loops on the FPGA. This enables easier monitoring of control loop execution delay as detailed in Section 8.6. The two loops communicate using two boolean status signals: ‘go’ which is asserted by the sequencing block logic to fire the control loop, and ‘done’ which is asserted by the control loop when processing is complete.

The clock generator block is configurable from the HMI PC and receives several inputs which can be selected as sample clock sources. In sampling-driven operation, the sample clock can be generated from the 40 MHz local FPGA clock or the external *PXI_Clk10* clock provided by the PXIe system. Both clock sources are routed via a programmable clock divider which is used to reduce the frequency to the desired sample clock frequency, and optionally offset the clock phase relative to the sample clocks of other control loops.

In event-driven operation, the clock generator block monitors control loop input values (eg. DAQ measurements or reference inputs from higher-level control loops) and generates a trigger event when $x[k] \neq x[k-1]$. The inputs monitored for value change events are dictated by the associated control loop and are detailed in Section 8.5.2.

Selecting the *PXI_Clk10* signal as a base clock in sampling-driven mode automatically ensures control loop phase alignment with minimal jitter (less than 50 ns) for any control loops which are configured to use this clock source. The operation of the *PXI_Clk10* clock divider block is described in the following section.

8.5.1 PXI_Clk10 Clock Divider Block Operation

As outlined previously, a design objective of the CHP control software rewrite was to implement clock division, phase offsetting, and global synchronisation of *PXI_Clk10*-derived control loop sample clocks on the PCU and ACU FPGAs. The operation of the *PXI_Clk10* clock divider block is described in detail in this section.

The architecture of the *PXI_Clk10* clock divider block is shown in Figure 8.4. The *PXI_Clk10* clock signal is received by the PCU and ACU FPGAs via the PXIe system backplane as shown previously in Figure 8.1 and Figure 8.2. The DAQ *aiStartTrigger* signal is routed onto the PXI trigger bus (*PXI_Trig[5]*) in the PXIe system by the DAQ card; this signal route is configured immediately by the LabVIEW VI running on the CCU when execution is started and before converter start-up. The *aiStartTrigger* signal is asserted (pulsed) once by the DAQ card when an analogue input measurement acquisition is started and is de-asserted (held low) at all other times. In the DAQ configuration used on the CCU, analogue input measurement acquisition is started programmatically by the LabVIEW VI during the CHP start-up routine. This means that all *PXI_Clk10* clock dividers are reset simultaneously during converter start-up, ensuring phase alignment.

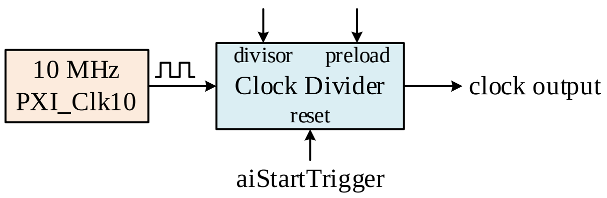


Figure 8.4: *PXI_Clk10* clock divider block architecture

The relationship between the global *aiStartTrigger* and *PXI_Clk10* signals, and the local FPGA clock signals is shown in the timing diagram in Figure 8.5. Timing information is shown at the top of the diagram in units of clock edges of the FPGA 40 MHz local clock.

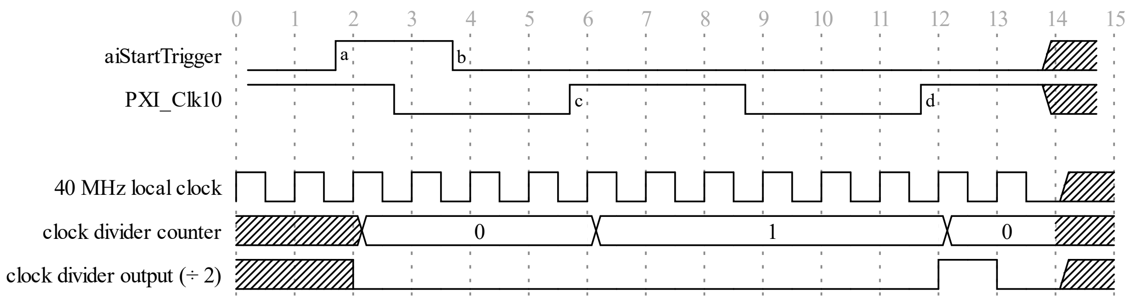


Figure 8.5: Timing diagram of global PXIe system synchronisation signals and PCU/ACU FPGA internal signals

As can be seen in Figure 8.5, *aiStartTrigger* and *PXI_Clk10* are asynchronous (ie. not phase aligned) to the 40 MHz FPGA local clock domain. These signals are translated into the 40 MHz clock domain by synchronisation registers in the FPGA.

Clock Divider Reset and Synchronisation Procedure

Inspecting the clock divider reset procedure first, it can be seen that *aiStartTrigger* is asserted by the DAQ card shortly before $t = 2$ at point **a** on the timing diagram. This is registered by the PCU or ACU FPGA on the next rising edge of the 40 MHz clock at $t = 2$, at which point the clock divider counter register is reset to zero and the clock divider output is de-asserted regardless of its previous state. Whilst in the reset state, the clock divider ignores any clock edges on the *PXI_Clk10* signal. It is important to note that the duration of the *aiStartTrigger* pulse is not provided by the NI PXIe-6363 DAQ device specifications, so the pulse duration shown in Figure 8.5 is only an estimate.

The *aiStartTrigger* signal is de-asserted by the DAQ card between $t = 3$ and $t = 4$ at point **b** on the timing diagram. This is registered by the PCU or ACU FPGA on the next rising edge of the 40 MHz clock at $t = 4$, at which point the clock divider is released from reset and will begin detecting clock edges on the *PXI_Clk10* input signal.

Clock Divider Operation

The required clock divisor to achieve the target control loop sample clock frequency is calculated by the HMI PC LabVIEW VI using Equation 8.1. The clock divisor is then passed to the corresponding clock divider block for that control loop on PCU or ACU FPGA. The counter value at which the count is reset is calculated by subtracting 1 from the clock divisor value.

$$\text{clock divisor} = \text{round}\left(\frac{\mathit{PXI\_Clk10}\,\text{frequency (Hz)}}{\text{target sample clock frequency (Hz)}}\right) \qquad 8.1$$

The clock divider counter is implemented using an unsigned 32-bit integer (`uint32`) register, therefore only integer clock divisor values in the `uint32` range (1 to 429,4967,295) are possible. As a result, there may be a small error in the output clock frequency with certain target sample clock values; the achieved sample clock after rounding is also shown on the HMI PC user interface. A clock divisor value of 0 is invalid and disables the clock divider output.

Upon exiting the reset state, the clock divider block begins counting rising edges on the *PXI_Clk10* signal. In the example shown in Figure 8.5, a clock divisor of 2 is used to generate a 5 MHz sample clock from the *PXI_Clk10* signal. The first *PXI_Clk10* clock edge after reset occurs shortly before *t* = 6 at point **c** on the diagram. This is registered by the FPGA at *t* = 6, at which point the clock divider counter is incremented. The next *PXI_Clk10* edge occurs at point **d** and the counter is incremented again. At this point the counter is reset since it has reached the value of: $\text{clock divisor} - 1$, and the clock divider output is pulsed for one cycle of the 40 MHz FPGA clock.

Clock Divider Output Phase Offset

To allow phase-offsetting of sample clocks relative to one another, the clock divider block provides a 'preload' input as shown in Figure 8.4. Using this terminal, the starting phase of the clock divider output can be offset in multiples of the *PXI_Clk10* clock period (100 ns).

The preload value is added to the maximum counter value at which the clock divider counter resets for the output clock cycle following a reset. This operation is shown in Figure 8.6 for a clock divisor of 2 and a preload value of 2. This equates to a clock divider output frequency of 5 MHz and a phase offset of 200 ns.

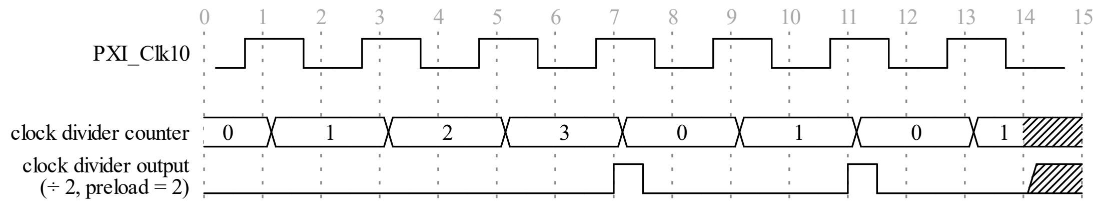


Figure 8.6: Timing diagram showing operation of the clock divider block preload input terminal

At *t* = 0, it is assumed that the clock divider has just been released from reset by the *aiStartTrigger* signal. As can be seen, the first pulse on the clock divider output occurs at *t* = 7 after four clock edges on *PXI_Clk10*, due to the addition of the preload value to the counter reset value. As a result, the starting phase of the clock divider output is offset. Subsequent clock pulses on the clock divider output occur at a counter value of 1, as expected for a clock divisor of 2.

The 40 MHz local FPGA clock divider blocks as shown in the following sections operate in a similar way to the *PXI_Clk10* clock divider blocks and are instead clocked directly by the 40 MHz clock rather than an external clock source. Furthermore, the reset and preload terminals are not used, since it is not possible to synchronise or phase offset control loop sample clocks generated using a local clock source.

8.5.2 Event-Driven Mode Value Change Event Sources

As stated previously, the inputs monitored by the clock generator block for value change events in event-driven sample clock mode is dictated by the associated control loop. These are outlined in the following sub-sections for each control loop implemented on the PCU and ACU, with reference to the cascaded control structure described in Section 2.2.

External Voltage Reference Phase-Locked Loop

The PLL for the external voltage reference input to the CHP is implemented on the PCU and calculates the phase angle from the 3-phase external voltage reference measurements taken by the DAQ card. In sampling-driven mode, the PLL can be configured to use the *PXI_Clk10* or 40 MHz FPGA local clocks as a sample clock source; this is the same for all control loops. In event-driven mode, the PLL can be triggered by a value change on any phase voltage in the 3-phase reference input ($V_{ext(abc)}$). The clock source arrangement for the PLL is shown in Figure 8.7.

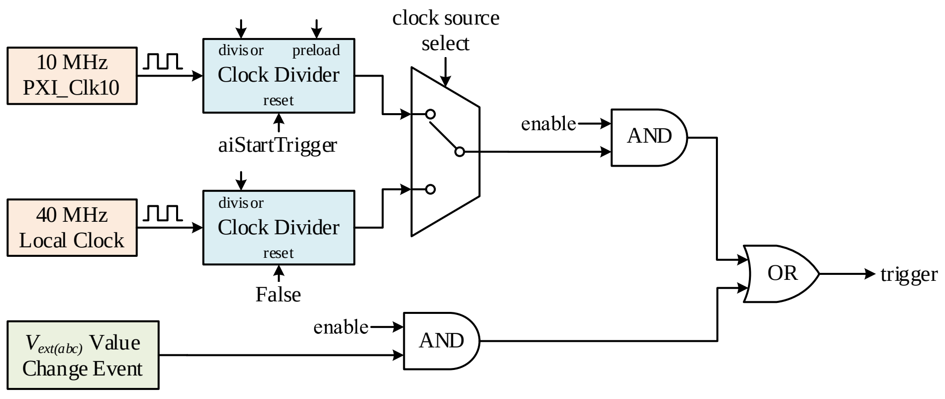


Figure 8.7: External reference voltage PLL sample clock sources

Circulating Current Suppression Control

The CCSC loop is implemented on the PCU and uses the six arm current measurements as an input to the control loop. The CCSC loop in the CHP is implemented in the dq reference frame; conversion from 3-phase abc quantities to dq and vice-versa requires the phase angle (θ) measurement from the PLL. As a result, in event-driven mode, the CCSC loop can be triggered on either a change in one or more of the six arm current measurements or PLL phase angle. This arrangement is shown in Figure 8.8.

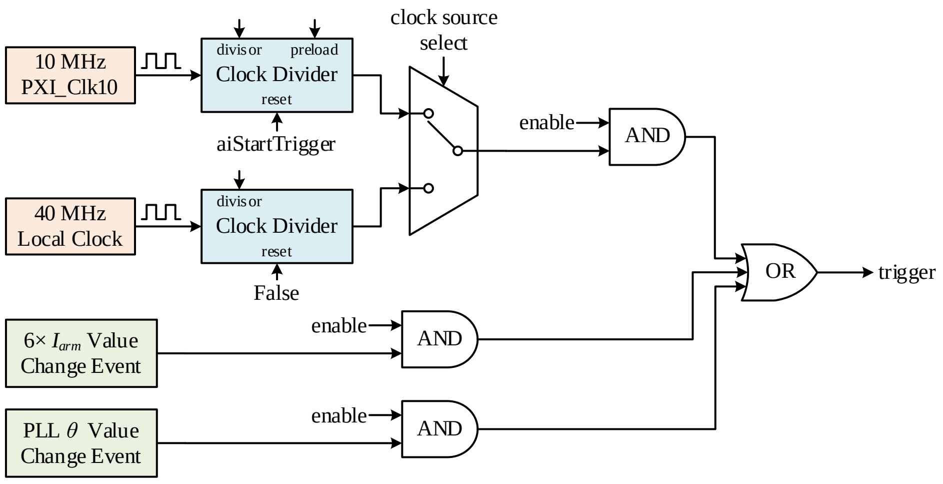


Figure 8.8: Circulating current suppression control loop sample clock sources

Output Voltage Control (External Voltage Reference)

The output voltage control loop for the CHP external voltage reference is implemented on the PCU and like the CCSC loop is implemented in the dq reference frame. As shown in Figure 8.9, the output voltage control loop can be triggered from a value change on $V_{ext(abc)}$ or PLL phase angle.

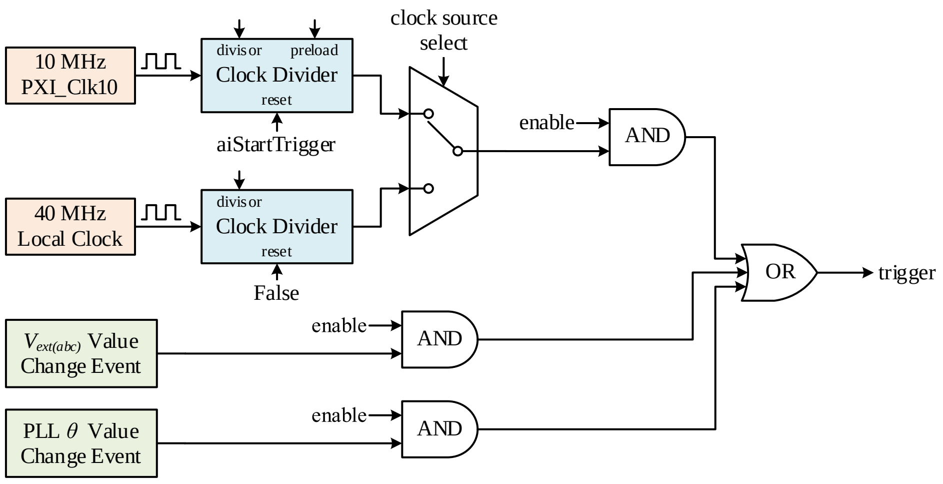


Figure 8.9: Output voltage control loop sample clock sources

Arm Balancing and Output Modulation (Nearest Level Control)

Arm balancing control and output modulation (nearest level control) are implemented on the PCU in the CHP. The arm balancing control loop calculates the six arm SM insertion indices (N_{on}) from the 3-phase AC output voltage set points from the output voltage controller or internal voltage reference generator ($V^*_{s(abc)}$) and the 3-phase difference voltage references ($V^*_{diff(abc)}$) from the CCSC loop. These inputs can be used as trigger sources in event-driven sampling mode, as shown in Figure 8.10.

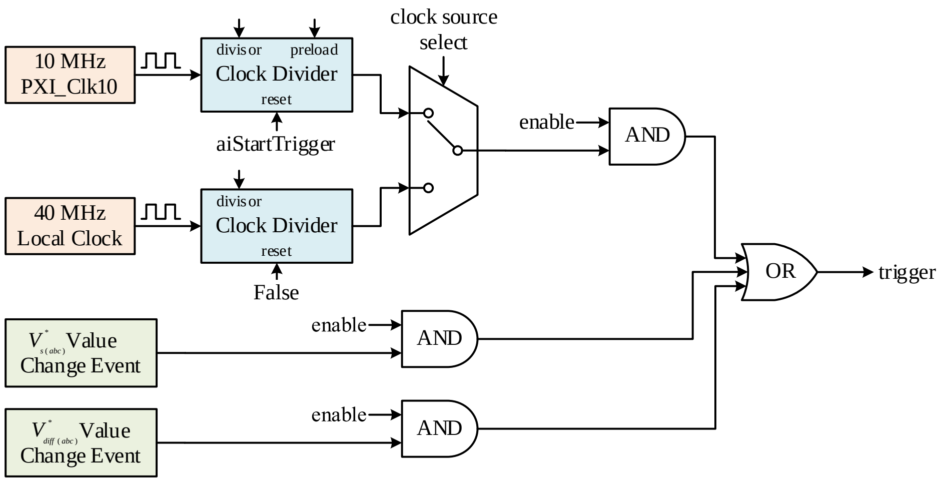


Figure 8.10: Arm balancing and NLC sample clock sources

Capacitor Balancing Control

The capacitor balancing control loop on the CHP is implemented on the ACU as traditional periodic (ie. sampling-driven) CBC. As a result, the CBC loop cannot be triggered by value change events on any inputs and operates only in sampling-driven mode, as shown in Figure 8.11.

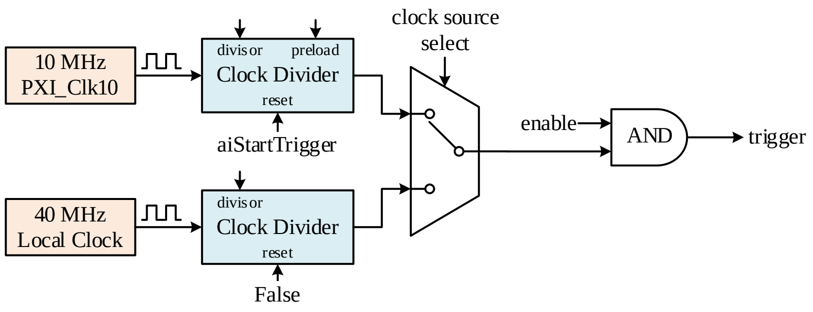


Figure 8.11: Capacitor balancing control loop sample clock sources

Firing Signal Generation

The firing signal generation loop on the CHP is implemented on the ACU and uses the six arm insertion indices (N_{on}) received from the PCU and the ranked list of SM identifiers from the CBC loop to determine which SMs to insert or bypass. In event-driven mode, the firing signal generator can be triggered by a value change event on any of the six insertion indices or a change in SMID ranking from the CBC loop. This arrangement is shown in Figure 8.12.

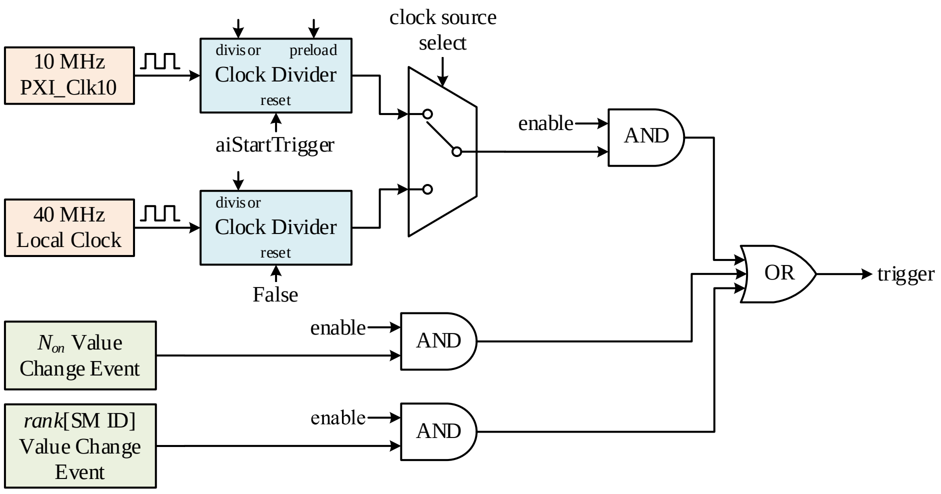


Figure 8.12: Firing signal generation loop sample clock sources

8.6 Hardware Delay Measurement

To enable measurement of data acquisition and control loop execution delay and phase alignment across DAQ, PCU, and ACU hardware targets, an external Digilent Digital Discovery PC-based digital logic analyser [107] is used. This device was chosen following a review of logic analyser hardware since it has a high maximum sample rate (800 MSa/s) and 24 input channels which make it suitable for measuring multiple timing signals across DAQ, PCU, and ACU with high precision.

For hardware delay measurements, the *go* and *done* signals from the sequencing block (shown in Figure 8.3) for control loops of interest are exported to digital output terminals on the PCU or ACU cards. These are then connected to the logic analyser inputs using a breakout board and jumper wires to provide easy re-configuration of output-input mappings. Any other internal signal of interest from the DAQ, PCU or ACU can also be routed to a digital output if required for timing measurements. For example, these may include the *trigger* signal, sample clocks, or the *PXI_Clk10* clock signal. The experimental set up for hardware delay measurements is shown in Figure 8.13.

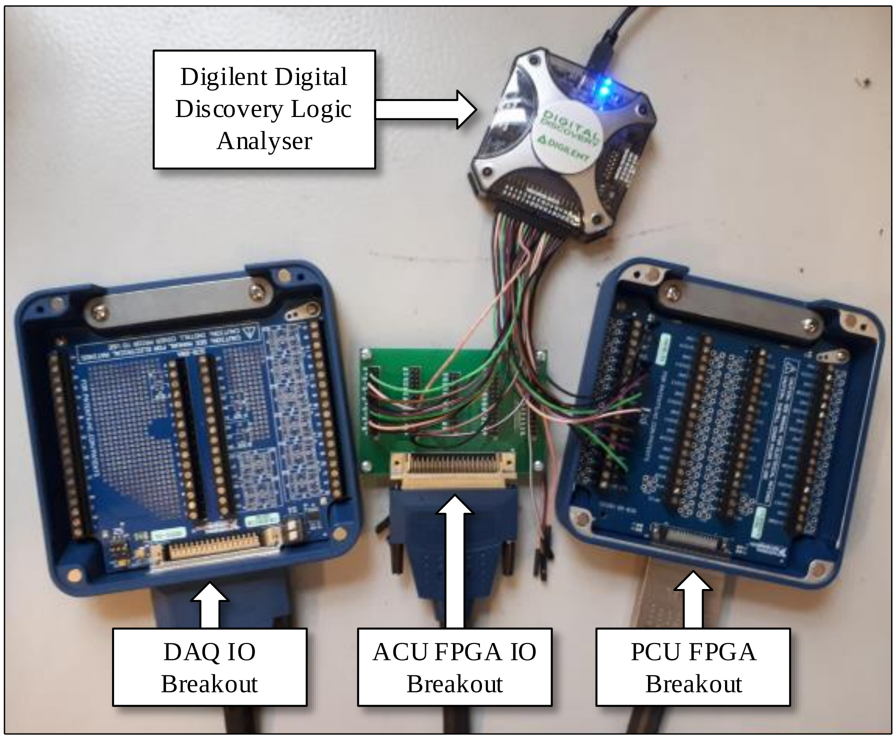


Figure 8.13: CHP control loop hardware delay measurement set up

An example logic analyser timing plot obtained from the NLC loop of the CHP whilst running is shown in Figure 8.14, validating the operation of the hardware timing measurement set up. As can be seen, the NLC loop takes 1.05 µs to execute as shown by the measurement cursors placed on the *NLC_Done* signal.

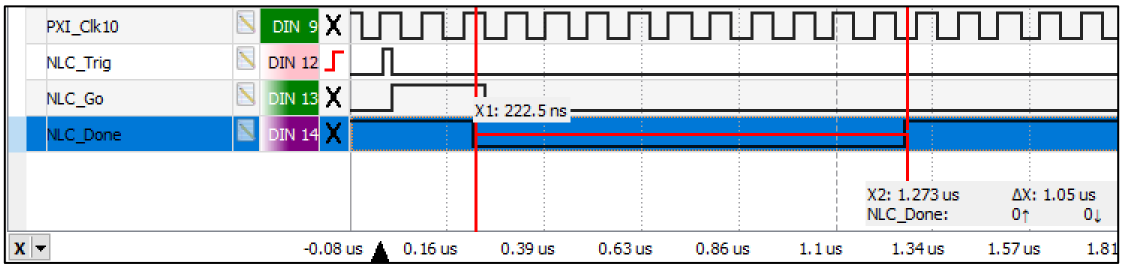


Figure 8.14: NLC loop timing data acquisition from logic analyser

8.7 Data Acquisition and Logging

As stated in the design objectives, the control software implementation should provide the ability to plot and/or log to file CHP internal measurements to increase observability. Measurements of $V_{ext(abc)}$, $I_{ac(abc)}$, I_{dc}, I_{arm}, and SM capacitor voltage are all available internally to the control software running on either the CCU, PCU or ACU – the aim was to extract this data and transfer it to the HMI PC for plotting and logging.

8.7.1 ACU Capacitor Voltage Data Transfer to CCU

The ACU receives forty-eight 10-bit SM capacitor voltage measurements from the twelve local controllers each time a new firing signal command packet is sent from the ACU to the LCs, typically at 20 kHz. Capacitor balancing in the CHP is typically performed at a lower sample rate in the order of 3 to 5 kHz, so capacitor voltage samples are transferred from ACU to CCU using this lower sample rate to reduce the load on the CPU in the CCU.

Capacitor voltage data transfer from the ACU to the CCU is implemented using a unidirectional direct memory access (DMA) first-in first-out (FIFO) buffer on the ACU FPGA. This is a standard LabVIEW FPGA construct for data transfer. The DMA buffer allows the ACU FPGA to write capacitor voltage samples directly to random-access memory (RAM) on the CCU, without requiring involvement of the CPU, freeing it up for other tasks. Data transfer into RAM is carried out across the PCI bus in the PXIe system.

When a new set of SM capacitor voltage samples are received by the ACU, the LabVIEW FPGA VI packs the samples into a packet, along with timestamp information (see Section 8.7.4) and places it in the FIFO buffer. The LabVIEW VI running on the CCU continuously monitors the number of elements in the FIFO buffer, and when a new set of samples is received (ie. the buffer is not empty), parses the sample data and re-packages it for transmission to the HMI PC as described in the following section.

8.7.2 Data Transfer to HMI PC

Capacitor voltage sample data and DAQ card measurements ($V_{ext(abc)}$, $I_{ac(abc)}$, I_{dc}, I_{arm}) are gathered by the LabVIEW VI running on the CCU before being transmitted over the LAN to the HMI PC for plotting and logging. The CCU VI packs sample data into packets, then sends the data over the LAN using the Network Stream Interface writer functionality provided in LabVIEW. Using the Network Stream Interface allows for lossless transfer of large amounts of data without requiring knowledge of the underlying transfer protocol.

8.7.3 Data Plotting and Logging on HMI PC

Packets containing sample data are received by the HMI PC using a Network Stream reader endpoint, before being unpacked for plotting and logging to file. All measurement data is plotted in real-time on the HMI PC front panel user interface and is also logged to a file on the HMI PC for post-processing and analysis. A screen capture of the HMI PC VI showing the SM capacitor voltage data plot using the data transferred from the ACU over the DMA and Network Stream interfaces is shown in Figure 8.15.

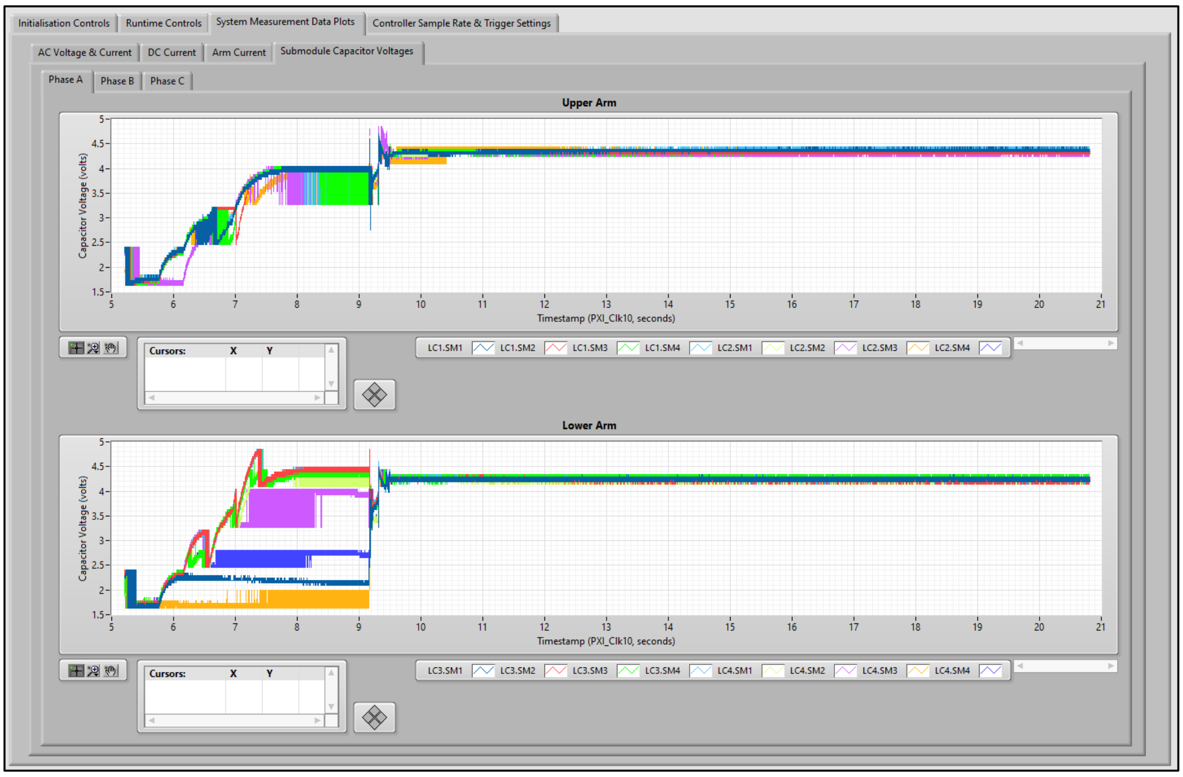


Figure 8.15: Screen capture of HMI PC user interface showing CHP SM capacitor voltage plots using data transferred from ACU FPGA

8.7.4 Measurement Timestamping

Since measurement samples are gathered from different hardware targets in the CHP (DAQ and ACU), a means of globally timestamping sample data across DAQ, PCU, and ACU was required to ensure that measurement data can be time-aligned for off-line analysis.

To provide this functionality, frequency counters are implemented on each of the DAQ, PCU, and ACU driven by a common clock signal (*aiSampleClock*) which is routed by the DAQ card onto the PXIe trigger bus as shown in Figure 8.2. The frequency counter blocks count each rising edge on *aiSampleClock* and store the count in a register which is readable by other parts of the LabVIEW FPGA VI in the case of the PCU and ACU, or by the CCU LabVIEW VI in the case of the DAQ card frequency counter.

When a new set of sample data is received by DAQ, PCU, or ACU, the current value of the frequency counter is read and stored as a timestamp along with the sample data. Since the frequency of *aiSampleClock* is known, the raw counter value can be converted to a timestamp in seconds using Equation 8.2. This calculation is carried out by the LabVIEW VI running on the CCU before the sample data and timestamp are packed for transmission to the HMI PC.

$$\text{timestamp (seconds)} = \text{counter value } \times \frac{1}{\mathit{aiSampleClock}\text{ frequency (Hz)}} \qquad 8.2$$

Synchronisation of all three counters is achieved using the *aiStartTrigger* signal in a similar way as used to synchronise the control loop sample clock generators. During converter start up, *aiStartTrigger* is asserted (pulsed) once by the DAQ card when analogue measurement acquisition starts. This resets all three counters to zero, providing a common $t = 0$ point for all measurements.

8.8 Code Optimisation

In addition to implementing the functionality required by the design objectives, the control software rewrite also provided the opportunity to optimise the LabVIEW code running on all platforms (HMI PC, CCU, PCU, and ACU). In particular, the optimisation process provided a reduction in FPGA logic resource usage on the ACU FPGA, despite the additional functionality added to the FPGA code. The logic resource usage for the PCU FPGA was slightly higher than the existing implementation, however, as with the ACU FPGA, additional functionality was added LabVIEW VI. A summary of the FPGA logic resource usage for the existing and new implementations is shown in Table 8.2.

Platform	**Existing Implementation** Total Slice Usage	**New Implementation** Total Slice Usage
PCU FPGA (PXIe-7857R)	11328 of 25350 (44.7 %)	11968 of 25350 (47.2 %)
ACU FPGA (PXI-7851R)	4670 of 4800 (97.3 %)	4561 of 4800 (95 %)

Table 8.2: CHP LabVIEW FPGA control software FPGA slice resource usage summary

Lower, or similar FPGA slice resource usage was achieved in part by code optimisation and by leveraging LabVIEW FPGA-specific constructs which have a more direct mapping to the underlying architecture of the FPGA. For example, the local controller UART transmitter and receiver blocks on the ACU were entirely re-written to use LabVIEW FPGA-specific constructs such as single-cycle timed loops (SCTL) and hand-coded finite-state machines (FSM). The re-written UART code was fully validated using the local controllers and fibre optic breakout board installed inside the CHP.

In addition to making extensive use of SCTLs and FSMs in the new CHP LabVIEW FPGA code, the method of passing data to different sections of code in the LabVIEW FPGA VI was fundamentally changed from the existing software implementation. In the existing code, extensive use was made of global variables to pass data between internal functional blocks (such as separate while loops) on the FPGA VIs. In many places, these global variables had multiple writers; this causes the LabVIEW FPGA compiler to automatically insert resource-intensive arbitration logic to ensure that only a single writer can update a variable stored in a register at any one time. The arbitration logic will also introduce jitter and non-determinism when updating variables, since several writers may be contesting to write to the same variable.

In the new control software implementation, the register construct in LabVIEW FPGA was used to replace global variables. Registers in LabVIEW FPGA have a direct mapping onto the fundamental memory storage elements in an FPGA and are a low-level construct. Furthermore, the dataflow in the VI was modified so that each register has only a single writer, to prevent the insertion of arbitration logic. Use of registers in place of global variables led to a significant reduction in like-for-like logic resource usage when the logic resources consumed by the additional functionality on the PCU and ACU FPGAs is accounted for.

8.9 Summary

This chapter has provided a detailed description of the new control system software written for the CHP during this project. The design objectives required to support the research during this project were identified, in addition to the limitations of the existing control software implementation. The methods used in the new software implementation to meet the objectives was then documented for each objective in turn, with a detailed description of the software components which provide the added functionality. The hardware results shown in Sections 8.6 and 8.7.1 validate the correct operation of the functionality added to the control software. By adding the ability to measure timing signals using external hardware, the execution time of each control loop running on the CHP can be measured and benchmarked against different control algorithm implementations. Furthermore, by using the hardware outputs, the synchronisation between control loops on the same or different FPGAs in the PXIe chassis can be measured accurately.

The specific details of the control software provided in this chapter are particularly applicable to distributed control and measurement systems used in CHPs in an academic setting, both in terms of scale (number of levels, converter ratings) and similarity of control hardware platforms. For industrial scale MMCs, the implementation specifics will differ due to scale; for example, control hardware platforms with greater processing power will be required, and communication links between control hardware targets will differ. Despite these differences, this chapter highlights several important factors which must be addressed in a distributed control system for an MMC operating at any scale. The implementation techniques developed in this work can be used in other converters to ensure accurate timing and synchronisation across distributed control systems in other MMCs. The most recent version of the control software developed during this project is available in the online supplementary repository [26].

9 Conclusion and Future Work

In this chapter, the main outcomes of the research will be summarised. Following this, opportunities for future work which has been identified during the project will be discussed.

9.1 Conclusion

The aim of this research was to provide a detailed insight into internal control delays and controller implementation techniques for an MMC, to guide hardware and software development, and improve simulation model fidelity by incorporating hardware delays into simulation models. To achieve this aim, the following objectives were identified:

1. Review and categorise CBC methods for an MMC and identify constraints on CBC loop execution delay.
2. Measure and compare the **resource usage** of a selection of sorting algorithms for CBC implemented on a range of industrially representative control hardware.
3. Measure and compare the **execution delay** of a selection of sorting algorithms for CBC implemented on a range of industrially representative control hardware.
4. Develop and validate a new suite of control software for the reduced-scale CHP which is suitable for investigating control loop delay and synchronisation.

All of these objectives have been met in this thesis, and in some cases further work has been undertaken.

The capacitor balancing control loop was selected for study since it plays a key role in controlling the internal dynamics of an MMC. Unbalanced capacitor voltages present a potential source of unreliability within an MMC, due to premature aging or over-voltage of SM components. Specifically, the sorting algorithm chosen for use in the CBC loop can introduce a potentially large and non-deterministic delay, which can lead to capacitor voltages exceeding the specified limits. Furthermore, the delay due to the sorting algorithm may dictate the maximum sampling frequency of the CBC loop, and therefore lead to constraints on other low-level converter control loops.

Following a review and classification of capacitor balancing control methods in Chapter 3, three methods were chosen for further study which allowed the investigation of sorting algorithm execution time with both fixed and variable CBC loop sampling periods. The review of sorting algorithms in Chapter 4 identified several sorting algorithms of interest based upon the existing literature: bubble sort, insertion sort, merge sort, quick sort, bitonic merge sort, and odd-even merge sort. These algorithms were implemented using three industrially representative programming languages (NI LabVIEW, LabVIEW FPGA, and Verilog) on two industrially representative control hardware platforms: a CPU running a real-time operating system, and a FPGA.

The execution time range measurements in Chapter 6 showed that the execution time ranges of merge sort and quick sort were several orders of magnitude higher than the other sorting algorithms. Coupled with the fact that these algorithms are recursive and cannot be easily implemented on an FPGA, which is the typical control hardware target for implementing the CBC loop, they were not analysed further and cannot be recommended for CBC methods requiring a fully sorted list of SM capacitor voltages. These results also showed that the execution time of the sorting networks (bitonic merge sort and odd-even merge sort) was fixed and consistently lowest across all implementation methods. This is a desirable characteristic when seeking to minimise the effects of delay and non-determinism upon capacitor voltage balancing.

The remaining sorting algorithms were then exercised using synthetic capacitor voltage data generated by the simulation model described in Chapter 5. To simplify the analysis, a delay threshold of 8 control cycles was applied to the measured execution times, since it was determined that capacitor balancing would typically only begin to degrade with a delay greater than 8 control cycles. Comparing the different combinations of sorting algorithm and CBC method across all simulation scenarios, insertion sort consistently out-performed bubble sort in terms of execution time. It can therefore be recommended where a classical sequential sorting algorithm is chosen for CBC. Comparing the different implementation methods (Verilog, LabVIEW FPGA Optimised, and LabVIEW FPGA Standard), it was seen that sorting algorithms implemented in Verilog and LabVIEW FPGA Optimised exhibited the lowest execution times. This is at the expense of increased implementation difficulty when compared to LabVIEW FPGA Standard.

The FPGA logic resource usage of each sorting algorithm and implementation method was then measured. This showed that, whilst both sorting networks exhibit low and fixed execution times, this comes at the expense of very high logic resource usage. Sorting algorithms implemented in Verilog and LabVIEW FPGA Optimised were shown to have lower logic resource usage than the equivalent implementation in LabVIEW FPGA Standard. As a result, these programming methods can be recommended where there is a need to optimise a design to fit on a selected FPGA device. In particular, the LabVIEW FPGA Optimised implementation route provides a good balance between sorting algorithm execution time, logic resource usage and implementation difficulty, so can be recommended in most cases.

The results from Chapter 6 can be used to guide the selection of a sorting algorithm for the CBC loop. Furthermore, the results show the direct link between CBC method sampling period and sorting algorithm execution time and how this must be accounted for in the control software development process. Where the sorting algorithm execution time delay is greater than 8 control cycles, the delay should be incorporated into electromagnetic transient simulation models to ensure that the simulation model reflects the behaviour of the CBC loop in hardware.

A new suite of control software was developed for the reduced-scale CHP MMC, as described in Chapter 8. The new control software implementation meets the desired objectives and will facilitate future research using the CHP, by allowing observation of internal measurements and runtime configuration of control loop sampling frequency and phase alignment.

The techniques used to implement controller synchronisation, phase alignment, and measurement timestamping in the new control software are highly applicable to cascaded, distributed control systems as used in other laboratory prototypes and industrial converters. Whilst the chosen control hardware target may differ, similar features such as shared trigger lines and clock signals will typically be available. These can be used in as described in in Chapter 8 to implement and benchmark control loops running on real-time digital control hardware. The considerations required when implementing control systems for laboratory prototype multilevel converters have been written up and published in journal format.

9.2 Future Work

Throughout the course of this work, several avenues for future research have been identified. These have been summarised in two categories as follows: capacitor balancing control loop and converter hardware prototype.

9.2.1 Capacitor Balancing Control Loop

Chapter 6 presented an in-depth analysis of the interaction between the choice of CBC method, sorting algorithm, and sorting algorithm execution time. In this work, the synthetic capacitor voltage data generated by the simulation model was sorted offline by the chosen sorting algorithm to generate the execution time data. Since sorting was performed offline, the sorting algorithm execution delay was not incorporated into the CBC loop in the simulation model; it was assumed to execute instantaneously, without delay. As a result, it was not possible to measure the per-timestep effect of sorting algorithm execution delay upon capacitor voltage deviation. The work carried out to select a threshold number of cycles delay presented in Chapter 6 and Appendix C does however prove that a delay within the CBC loop will degrade capacitor voltage balancing performance, so this area warrants further research.

The next stage of investigation into sorting algorithms for CBC is to incorporate real-world sorting algorithm execution delay measurements into the simulation model. To do this, it will be necessary to build upon the PSCAD/EMTDC processor-in-the-loop component presented in [108]. Using this component, capacitor voltage data can be sent to the control hardware target (such as a CPU or FPGA) for sorting at each simulation timestep. The real-world execution time measurement is then returned to the simulation, along with the sorted capacitor voltage data. The received data must then be placed into some form of first-in first-out (FIFO) queue, which only releases the data to the downstream modulation algorithm after the execution time delay has elapsed. The delayed FIFO queue will need to be implemented using a PSCAD/EMTDC custom component; it may be possible to follow a similar approach to that presented in [109].

The results from this work could be used to provide a more detailed insight into capacitor balancing behaviour in the presence of sorting algorithm execution delay. In particular, the cumulative effect of delays in previous CBC loop cycles can be measured, along with the effect of variable (non-deterministic) delays, typical of classical sorting algorithms.

A similar approach to that detailed above can also be used to incorporate other delays into the CBC loop, such as communication network delays. Furthermore, this technique could be used to investigate the behaviour of other control loops in the presence of non-zero, non-deterministic processing delays. The results from this work could lead to more accurate simulation models, and the development of new simulation techniques which link offline simulation in PSCAD/EMTDC with control hardware operating in real-time. These simulation techniques could be applied to a wide range of power electronic-based converters, or indeed any simulation model which can be built in PSCAD/EMTDC.

Execution Time-Limited Sorting Algorithms

In this work, it is assumed that it is necessary for the sorting algorithm to finish executing and produce a fully sorted list of SM capacitor voltages for proper operation of the CBC loop. As has been shown, this requirement places an upper limit on sorting algorithm execution time, which is often exceeded by the sorting algorithm.

A piece of future work could involve removing the requirement for the sorting algorithm to finish executing before the CBC loop can proceed. Instead, the CBC loop could force termination of the sorting algorithm as soon as the list of SM identifiers are required by the modulation algorithm. The CBC loop then retrieves the working array of SM capacitor voltages and identifiers operated on by the sorting algorithm in its current state, which may be partially sorted.

The effectiveness of this approach is likely to be highly dependent upon the chosen sorting algorithm. For example, some sorting algorithms such as insertion sort produce a sorted array which increases in length at each iteration – that is – the working array is never completely un-ordered. Were this approach to show that capacitor voltage balancing is not significantly affected by early termination of the sorting algorithm, this may be a means of overcoming the often large and non-deterministic delay inherent in many classical sorting algorithms.

9.2.2 Converter Hardware Prototype

Control Loop Delay and Synchronisation Investigation

The CHP hardware modifications and the control software re-write carried out during this project have opened up several areas of future research exploring delays and synchronisation in cascaded, distributed control systems. In the first instance, the ability to access and measure the *trigger* and *go*/*done* signals of internal control loops in the CHP using a logic analyser could be used to validate the execution time of the sorting algorithm for CBC as implemented on the CHP.

This work could be extended further by benchmarking the execution time of the other control loops on the CHP. This will provide a more detailed insight into the components which make up the CHP total system delay originally presented in [25]. These timing measurements could be incorporated into simulation models to improve their fidelity.

Following the benchmarking of the control loops on the CHP, an area of further work could involve investigating methods to minimise the total system delay. This can be done by adjusting the sampling rate and synchronisation of control loops relative to one another. This work would make use of the ability to configure control loop trigger sources, sampling rates and phase offsets relative to a primary clock source which has been implemented in the new control software. The results from this work could be used to develop a critical path of cascaded control and identify which control loops must be time aligned to produce the minimum total system delay.

Hardware and Software Development

Whilst the hardware modifications and control software re-write have resolved several issues with the CHP and facilitated the research carried out during this project, further development is always possible. Due to the complexity of the CHP hardware and control software, some issues are still outstanding, or have been found whilst validating the new control software. For example, logging of system measurements and SM capacitor voltages can only be enabled for operating runs of less than approximately 1 minute, due to poor memory management by the specific implementation of data plotting in the HMI LabVIEW VI.

Future hardware and software development may involve resolving these issues or modifying the CHP further to suit a new research objective. The reconfigurable, distributed control architecture of the CHP opens up a wide range of research avenues; these may include investigating multi-terminal DC networks or coordinated control of multiple VSCs across long distance telecommunication links.

References

[1] "The Climate Change Act 2008 (2050 Target Amendment) Order 2019," ed: UK Parliament, 2019.

[2] "New plans to make UK world leader in green energy," ed. United Kingdom: Prime Minister's Office, 10 Downing Street, 2020.

[3] "Offshore Wind Leasing Round 4 signals major vote of confidence in the UK's green economy." The Crown Estate. https://www.thecrownestate.co.uk/en-gb/media-and-insights/news/2021-offshore-wind-leasing-round-4-signals-major-vote-of-confidence-in-the-uk-s-green-economy/ (accessed 8th Jun., 2022).

[4] "ScotWind offshore wind leasing delivers major boost to Scotland's net zero aspirations." Crown Estate Scotland. https://www.crownestatescotland.com/news/scotwind-offshore-wind-leasing-delivers-major-boost-to-scotlands-net-zero-aspirations (accessed 8th Jun., 2022).

[5] R. Norris. "Offshore wind pipeline surges to 86 gigawatts, boosting UK's energy independence." RenewableUK. https://www.renewableuk.com/news/599739/Offshore-wind-pipeline-surges-to-86-gigawatts-boosting-UKs-energy-independence.htm (accessed 8th Jun., 2022).

[6] J. Lee *et al.*, "Global Offshore Wind Report 2021," Global Wind Energy Council, Brussels, Belgium, 9th Sep. 2021 2021. [Online]. Available: https://gwec.net/wp-content/uploads/2021/09/GWEC-offshore-wind-2021-updated-1.pdf

[7] J. Arrillaga, Y. H. Liu, and N. R. Watson, *Flexible Power Transmission: The HVDC Options*. Chichester, UK: John Wiley & Sons, Ltd, 2007.

[8] A. Alassi, S. Bañales, O. Ellabban, G. Adam, and C. MacIver, "HVDC Transmission: Technology Review, Market Trends and Future Outlook," *Renewable and Sustainable Energy Reviews,* vol. 112, pp. 530-554, 2019/09/01 2019, doi: https://doi.org/10.1016/j.rser.2019.04.062.

[9] "2020 Offshore Wind Project Timelines." RenewableUK. https://cdn.ymaws.com/www.renewableuk.com/resource/resmgr/ruk20_offshore_timeline_fina.pdf (accessed 8th Jun., 2022).

[10] "Offshore Wind Leasing Round 4 Selected Projects." The Crown Estate. https://www.thecrownestate.co.uk/media/3721/the-crown-estate-offshore-wind-leasing-round-4-selected-projects.pdf (accessed 8th Jun., 2022).

[11] "ScotWind List of Successful Project Partners." Crown Estate Scotland. https://www.crownestatescotland.com/resources/documents/scotwind-list-of-successful-project-partners-170122 (accessed 8th Jun., 2022).

[12] A. Gray. "Initial Predictions for Offshore Wind Farms in the ScotWind Leasing Round." ORE Catapult. https://ore.catapult.org.uk/wp-content/uploads/2021/03/AIPaper_ScotWind_Predictions_FINAL.pdf (accessed 8th Jun., 2022).

[13] "Offshore Transmission Network Review." UK Government. https://www.gov.uk/government/groups/offshore-transmission-network-review (accessed 8th Jun., 2022).

[14] "Renewable Curtailment and the Role of Long Duration Storage," Lane Clark & Peacock; Drax, United Kingdom, 2022. [Online]. Available: https://www.drax.com/wp-content/uploads/2022/06/Drax-LCP-Renewable-curtailment-report-1.pdf

[15] I. Staffell, R. Green, T. Green, R. Gross, and M. Jansen, "Drax Electric Insights Q4 2020 Report," Drax, United Kingdom, 2020. [Online]. Available: https://reports.electricinsights.co.uk/wp-content/uploads/2021/02/Drax-Electric-Insights-Q4-2020-Report.pdf

[16] G. Asplund, K. Eriksson, and K. Svensson, "DC Transmission based on Voltage Source Converters," presented at the CIGRE SC14 Colloquium, South Africa, 1997. [Online]. Available: https://library.e.abb.com/public/5afdef8c2ecc8394c1256fda004c8ca8/lite02.pdf.

[17] M. Barnes and A. Beddard, "Voltage Source Converter HVDC Links – The State of the Art and Issues Going Forward," *Energy Procedia,* vol. 24, pp. 108-122, 2012/01/01 2012, doi: https://doi.org/10.1016/j.egypro.2012.06.092.

[18] K. Sharifabadi, L. Harnefors, H.-P. Nee, S. Norrga, and R. Teodorescu, *Design, Control and Application of Modular Multilevel Converters for HVDC Transmission Systems*. Chichester, UK: John Wiley & Sons, Ltd, 2016.

[19] M. Barnes and T. Heath, "VSC-HVDC Newsletter: May 2022," ed. Manchester, UK, 2022.

[20] J. Glasdam, N. Qin, and V. Akhmatov, "Requirements and Quality Assurance of HVDC Digital Models for Power System Analysis," presented at the CIGRE Symposium Aalborg 2019, Aalborg, Denmark, 2019, 37.

[21] I. Jahn, M. Nahalparvari, S. Norrga, and R. Rogersten, "Moving Beyond Open-Source Modelling: Why Open Control and Protection Software in Real Converters Will Be Useful," in *2022 Open Source Modelling and Simulation of Energy Systems (OSMSES)*, 4-5 April 2022 2022, pp. 1-6, doi: 10.1109/OSMSES54027.2022.9769139.

[22] "Developing Open-Source Converter Models – The National HVDC Centre." The National HVDC Centre. https://www.hvdccentre.com/innovation-projects/open-source-converters/ (accessed 3rd May, 2022).

[23] T. Heath, P. R. Green, M. Barnes, and P. Coventry, Puerto Varas. Capacitor Balancing Controller Voltage Sorting Statistics in Modular Multilevel Converters.

[24] S. Yang, A. Bryant, P. Mawby, D. Xiang, L. Ran, and P. Tavner, "An Industry-Based Survey of Reliability in Power Electronic Converters," *IEEE Transactions on Industry Applications,* vol. 47, no. 3, pp. 1441-1451, 2011, doi: 10.1109/TIA.2011.2124436.

[25] T. Heath, "Modular Multilevel Converter Hardware and Simulation Comparison," Doctor of Philosophy, Department of Electrical and Electronic Engineering, The University of Manchester, Manchester, UK, 2019.

[26] J. Andrews. *Supplementary Online Repository for Computational and Communication Architectures for Modular Multilevel Converter Construction*, Mendeley Data, 2022, doi: https://dx.doi.org/10.17632/fr2jrff9w3.1.

[27] T. Qingrui, X. Zheng, H. Huang, and Z. Jing, "Parameter Design Principle of the Arm Inductor in Modular Multilevel Converter based HVDC," in *2010 International Conference on Power System Technology,* 24-28 Oct. 2010 2010, pp. 1-6, doi: 10.1109/POWERCON.2010.5666416.

[28] P. K. Bjorn Jacobson, Gunnar Asplund, Lennart Harnefors, Tomas Jonsson, "VSC-HVDC Transmission with Cascaded Two-Level Converters," in *CIGRE 2010*, Paris, 2010: CIGRE. [Online]. Available: https://library.e.abb.com/public/422dcbc564d7a3e1c125781c00507e47/B4-110_2010%20-%20VSC-HVDC%20Transmission%20with%20Cascaded%20Two-level%20converters.pdf. [Online]. Available: https://library.e.abb.com/public/422dcbc564d7a3e1c125781c00507e47/B4-110_2010%20-%20VSC-HVDC%20Transmission%20with%20Cascaded%20Two-level%20converters.pdf

[29] A. Nami, J. Liang, F. Dijkhuizen, and G. D. Demetriades, "Modular Multilevel Converters for HVDC Applications: Review on Converter Cells and Functionalities," *IEEE Transactions on Power Electronics,* vol. 30, no. 1, pp. 18-36, 2015, doi: 10.1109/TPEL.2014.2327641.

[30] Q. Tu, Z. Xu, and J. Zhang, "Circulating Current Suppressing Controller in Modular Multilevel Converter," in *IECON 2010 - 36th Annual Conference on IEEE Industrial Electronics Society*, 7-10 Nov. 2010 2010, pp. 3198-3202, doi: 10.1109/IECON.2010.5675048.

[31] A. J. Beddard, "Factors Affecting the Reliability of VSC-HVDC for the Connection of Offshore Windfarms," PhD, School of Electrical and Electronic Engineering, The University of Manchester, Manchester, UK, 2014.

[32] A. Antonopoulos, L. Angquist, and H. Nee, "On Dynamics and Voltage Control of the Modular Multilevel Converter," in *2009 13th European Conference on Power Electronics and Applications*, 8-10 Sept. 2009 2009, pp. 1-10.

[33] Q. Tu, Z. Xu, and L. Xu, "Reduced Switching-Frequency Modulation and Circulating Current Suppression for Modular Multilevel Converters," *IEEE Transactions on Power Delivery,* vol. 26, pp. 2009-2017, 2011, doi: 10.1109/TPWRD.2011.2115258.

[34] L. Harnefors, A. Antonopoulos, S. Norrga, L. Angquist, and H. Nee, "Dynamic Analysis of Modular Multilevel Converters," *IEEE Transactions on Industrial Electronics,* vol. 60, no. 7, pp. 2526-2537, 2013, doi: 10.1109/TIE.2012.2194974.

[35] S. Debnath, J. Qin, B. Bahrani, M. Saeedifard, and P. Barbosa, "Operation, Control, and Applications of the Modular Multilevel Converter: A Review," *IEEE Transactions on Power Electronics,* vol. 30, no. 1, pp. 37-53, 2015, doi: 10.1109/TPEL.2014.2309937.

[36] M. A. Perez, S. Bernet, J. Rodriguez, S. Kouro, and R. Lizana, "Circuit Topologies, Modeling, Control Schemes, and Applications of Modular Multilevel Converters," *IEEE Transactions on Power Electronics,* vol. 30, no. 1, pp. 4-17, 2015, doi: 10.1109/TPEL.2014.2310127.

[37] S. Wang, G. P. Adam, A. M. Massoud, D. Holliday, and B. W. Williams, "Analysis and Assessment of Modular Multilevel Converter Internal Control Schemes," *IEEE Journal of Emerging and Selected Topics in Power Electronics,* vol. 8, no. 1, pp. 697-719, 2020, doi: 10.1109/JESTPE.2019.2899794.

[38] A. Lesnicar and R. Marquardt, "An Innovative Modular Multilevel Converter Topology Suitable for a Wide Power Range," in *2003 IEEE Bologna Power Tech Conference Proceedings,* Bologna, 2003, vol. 3: IEEE, pp. 272-277, doi: 10.1109/PTC.2003.1304403.

[39] J. Qin and M. Saeedifard, "Reduced Switching-Frequency Voltage-Balancing Strategies for Modular Multilevel HVDC Converters," *IEEE Transactions on Power Delivery,* vol. 28, no. 4, pp. 2403-2410, 2013, doi: 10.1109/TPWRD.2013.2271615.

[40] A. António-Ferreira and O. Gomis-Bellmunt, "Comparison of Cell Selection Methods for Modular Multilevel Converters," presented at the International Conference on Environment and Electrical Enginerring, 2016.

[41] A. Hassanpoor, K. Ilves, S. Norrga, L. Angquist, and H. P. Nee, "Tolerance-Band Modulation Methods for Modular Multilevel Converters," presented at the 2013 15th European Conference on Power Electronics and Applications, EPE 2013, 2013.

[42] A. Hassanpoor, L. Ängquist, S. Norrga, K. Ilves, and H. Nee, "Tolerance Band Modulation Methods for Modular Multilevel Converters," *IEEE Transactions on Power Electronics,* vol. 30, no. 1, pp. 311-326, 2015, doi: 10.1109/TPEL.2014.2305114.

[43] A. Hassanpoor, "Modulation of Modular Multilevel Converters for HVDC Transmission," KTH Royal Institute of Technology, Stockholm, 2016. [Online]. Available: http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-192607

[44] A. Hassanpoor, A. Roostaei, S. Norrga, and M. Lindgren, "Optimization-Based Cell Selection Method for Grid-Connected Modular Multilevel Converters," *IEEE Transactions on Power Electronics,* vol. 31, pp. 2780-2790, 2016, doi: 10.1109/TPEL.2015.2448573.

[45] C. Lascu, E. Serban, C. Pondiche, and T. Dragicevic, "Anticipative Sorting Control of Modular Multilevel Converters," presented at the 2018 IEEE Energy Conversion Congress and Exposition, ECCE 2018, 2018.

[46] F. Deng and Z. Chen, "A Control Method for Voltage Balancing in Modular Multilevel Converters," *IEEE Transactions on Power Electronics,* vol. 29, pp. 66-76, 2014, doi: 10.1109/TPEL.2013.2251426.

[47] M. Saeedifard and R. Iravani, "Dynamic Performance of a Modular Multilevel Back-to-Back HVDC System," *IEEE Transactions on Power Delivery,* vol. 25, pp. 2903-2912, 2010, doi: 10.1109/TPWRD.2010.2050787.

[48] S. Rohner, S. Bernet, M. Hiller, and R. Sommer, "Modulation, Losses, and Semiconductor Requirements of Modular Multilevel Converters," *IEEE Transactions on Industrial Electronics,* vol. 57, no. 8, pp. 2633-2642, 2010, doi: 10.1109/TIE.2009.2031187.

[49] J. Qiu, T. Dai, Y. Cui, S. Xin, and Y. Lu, "Research on Capacitance Voltage Balance of Modular Multilevel Converter Based on Improved Quick Sorting Method," presented at the Proceedings - 2017 Chinese Automation Congress, CAC 2017, 2017.

[50] J. Zhang, J. Liu, J. Liu, W. Fang, J. Hou, and Y. Dong, "Modified Capacitor Voltage Balancing Sorting Algorithm for Modular Multilevel Converter," *The Journal of Engineering,* vol. 2019, pp. 3315-3319, 2019, doi: 10.1049/joe.2018.8910.

[51] T. Wang, H. Lin, Z. Wang, Y. Ma, and X. Wang, "A Fast Selection Algorithm Based on Binary Numbers for Capacitor Voltage Balance in Modular Multilevel Converter," presented at the 2018 IEEE Applied Power Electronics Conference and Exposition (APEC), 2018.

[52] D. Wang and B. Zhang, "A Optimized Capacitor Voltage Balance Strategy using Binary Number Sorting Algorithm for Modular Multilevel Converter," in *2019 8th International Symposium on Next Generation Electronics (ISNE)*, 9-10 Oct. 2019 2019, pp. 1-3, doi: 10.1109/ISNE.2019.8896591.

[53] W. Lin, W. Ping, L. Zixin, and L. Yaohua, "A Novel Capacitor Voltage Balancing Control Strategy for Modular Multilevel Converters (MMC)," in *2013 International Conference on Electrical Machines and Systems (ICEMS),* 26-29 Oct. 2013, pp. 1804-1807, doi: 10.1109/ICEMS.2013.6713283.

[54] D. Siemaszko, "Fast Sorting Method for Balancing Capacitor Voltages in Modular Multilevel Converters," *IEEE Transactions on Power Electronics,* vol. 30, pp. 463-470, 2015, doi: 10.1109/TPEL.2014.2312101.

[55] J. Mei, B. Xiao, K. Shen, L. M. Tolbert, and J. Y. Zheng, "Modular Multilevel Inverter with New Modulation Method and its Application to Photovoltaic Grid-connected Generator," *IEEE Transactions on Power Electronics,* vol. 28, pp. 5063-5073, 2013, doi: 10.1109/TPEL.2013.2243758.

[56] J. Mei, K. Shen, B. Xiao, L. M. Tolbert, and J. Zheng, "A New Selective Loop Bias Mapping Phase Disposition PWM With Dynamic Voltage Balance Capability for Modular Multilevel Converter," *IEEE Transactions on Industrial Electronics,* vol. 61, no. 2, pp. 798-807, 2014, doi: 10.1109/TIE.2013.2253069.

[57] S. Fan, K. Zhang, J. Xiong, and Y. Xue, "An Improved Control System for Modular Multilevel Converters Featuring New Modulation Strategy and Voltage Balancing Control," in *2013 IEEE Energy Conversion Congress and Exposition*, 15-19 Sept. 2013 2013, pp. 4000-4007, doi: 10.1109/ECCE.2013.6647231.

[58] W. Li, L.-A. Gregoire, and J. Belanger, "A Modular Multilevel Converter Pulse Generation and Capacitor Voltage Balance Method Optimized for FPGA Implementation," *IEEE Transactions on Industrial Electronics,* vol. 62, pp. 2859-2867, 2015, doi: 10.1109/TIE.2014.2362879.

[59] M. Hagiwara and H. Akagi, "Control and Experiment of Pulsewidth-Modulated Modular Multilevel Converters," *IEEE Transactions on Power Electronics,* vol. 24, pp. 1737-1746, 2009, doi: 10.1109/TPEL.2009.2014236.

[60] X. Li, Q. Song, J. Li, and W. Liu, "Capacitor Voltage Balancing Control Based on CPS-PWM of Modular Multilevel Converter," in *2011 IEEE Energy Conversion Congress and Exposition*, 17-22 Sept. 2011 2011, pp. 4029-4034, doi: 10.1109/ECCE.2011.6064317.

[61] J. Qin and M. Saeedifard, "Predictive Control of a Modular Multilevel Converter for a Back-to-Back HVDC System," *IEEE Transactions on Power Delivery,* vol. 27, no. 3, pp. 1538-1547, 2012, doi: 10.1109/TPWRD.2012.2191577.

[62] J. W. Moon, J. S. Gwon, J. W. Park, D. W. Kang, and J. M. Kim, "Model Predictive Control with a Reduced Number of Considered States in a Modular Multilevel Converter for HVDC System," *IEEE Transactions on Power Delivery,* vol. 30, pp. 608-617, 2015, doi: 10.1109/TPWRD.2014.2303172.

[63] A. Hassanpoor, A. Nami, and S. Norrga, "Tolerance Band Adaptation Method for Dynamic Operation of Grid-Connected Modular Multilevel Converters," *IEEE Transactions on Power Electronics,* vol. 31, pp. 8172-8181, 2016, doi: 10.1109/TPEL.2016.2521480.

[64] A. Hassanpoor, "A multilevel converter with cells being selected based on phase arm current," Patent Appl. PCT/EP2012/073699, 2012.

[65] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*. The MIT Press, 2009, p. 1312.

[66] M. Ricco, L. Mathe, E. Monmasson, and R. Teodorescu, "FPGA-Based Implementation of MMC Control Based on Sorting Networks," *Energies,* vol. 11, p. 2394, 2018, doi: 10.3390/en11092394.

[67] M. Ricco, L. Mathe, and R. Teodorescu, "FPGA-based Implementation of Sorting Networks in MMC Applications," presented at the 2016 18th European Conference on Power Electronics and Applications, EPE 2016 ECCE Europe, 2016.

[68] Q. Wang, K. Liu, K. Wang, L. Qin, J. Zhang, and Q. Pu, "Fast Capacitor Voltage Balancing Strategy Based on System History Operation Information for MMC," *IET Generation, Transmission & Distribution,* vol. 13, pp. 1104-1109, 2019, doi: 10.1049/iet-gtd.2018.6227.

[69] F. Zhao, G. Xiao, M. Liu, and D. Yang, "A Fast Sorting Strategy Based on a Two-way Merge Sort for Balancing the Capacitor Voltages in Modular Multilevel Converters," *Journal of Power Electronics*, vol. 17, pp. 346-357, 2017, doi: 10.6113/JPE.2017.17.2.346.

[70] F. Zhao, G. Xiao, Z. Song, and C. Peng, "Insertion Sort Correction of Two-way Merge Sort Algorithm for Balancing Capacitor Voltages in MMC with Reduced Computational Load," presented at the 2016 IEEE 8th International Power Electronics and Motion Control Conference, IPEMC-ECCE Asia 2016, 2016.

[71] D. Paradis, "Real-time Simulation of Modular Multilevel Converters," MSc, Electrical and Computer Engineering, University of Toronto, Toronto, US, 2013. [Online]. Available: https://tspace.library.utoronto.ca/handle/1807/43285

[72] X. Zhang, X. Kang, M. Guo, Y. Zhang, and Y. Xu, "An Optimization Algorithm of Capacitor Voltage Balancing in Modular Multilevel Converter," in *2018 International Conference on Power System Technology (POWERCON)*, 6-8 Nov. 2018 2018, pp. 2533-2537, doi: 10.1109/POWERCON.2018.8602343.

[73] D. E. Knuth, *The Art of Computer Programming Vol. 3: Sorting and Searching*, 3rd ed. Pearson Addison Wesley, 1998.

[74] K. E. Batcher, "Sorting Networks and Their Applications," presented at the Proceedings of the April 30-May 2, 1968, Spring Joint Computer Conference, Atlantic City, New Jersey, 1968.

[75] *Developing Open-Source Converter Models: Offline PSCAD/EMTDC Model*. (2019). The National HVDC Centre, University of Strathclyde. [Online]. Available: https://www.hvdccentre.com/wp-content/uploads/2019/12/PSCAD-Model.zip

[76] D. Guo, M. H. Rahman, G. P. Adam, and A. Emhemed, "Validation of Real-time User-defined MMC Models," 24th Aug. 2018, issue 3. [Online]. Available: https://www.hvdccentre.com/wp-content/uploads/2019/12/The-National-HVDC-Center-Project-Report-II-MMC-RTDS-Model.pdf

[77] D. Guo, M. H. Rahman, G. P. Adam, and A. Emhemed, "Offline DC Grid Model Development," 12th Sep. 2018, issue 3. [Online]. Available: https://www.hvdccentre.com/wp-content/uploads/2019/12/The-National-HVDC-Center-Project-Report-III-DC-Grid-Model.pdf

[78] D. Guo, M. H. Rahman, G. P. Adam, and A. Emhemed, "Development and Validation of Offline and Real-time User-defined Models of Alternative MMC Configurations," 26th Sep. 2018, issue 4. [Online]. Available: https://www.hvdccentre.com/wp-content/uploads/2019/12/The-National-HVDC-Center-Project-Report-IV-FB-and-Hybrid-MMC-Model.pdf

[79] D. Guo, M. H. Rahman, G. P. Adam, and A. Emhemed, "DC Grid with User-defined Converter Models: Validation of Real-Time Model in RTDS against Offline Equivalents in PSCAD/EMTDC Environment," 31st Oct. 2018, issue 5. [Online]. Available: https://www.hvdccentre.com/wp-content/uploads/2019/12/The-National-HVDC-Center-Project-Report-V-MTDC-Grid-RTDS-Model.pdf

[80] M. H. Rahman, D. Guo, G. P. Adam, and A. Emhemed, "Development and Validation of Offline and Real-time User-defined Models of Alternative MMC Configurations," 15th Apr. 2018, issue 2. [Online]. Available: https://www.hvdccentre.com/wp-content/uploads/2019/12/The-National-HVDC-Center-Project-Report-I-MMC-Modelling.pdf

[81] D. Ronanki and S. S. Williamson, "Failure Prediction of Submodule Capacitors in Modular Multilevel Converter by Monitoring the Intrinsic Capacitor Voltage Fluctuations," *IEEE Transactions on Industrial Electronics*, vol. 67, no. 4, pp. 2585-2594, 2020, doi: 10.1109/TIE.2019.2912771.

[82] T. Heath *et al.*, "Cascaded- and Modular-Multilevel Converter Laboratory Test System Options: A Review," *IEEE Access*, vol. 9, pp. 44718-44737, 2021, doi: 10.1109/ACCESS.2021.3066261.

[83] "NI PXIe-8101/8102 User Manual and Specifications - National Instruments." National Instruments. https://www.ni.com/pdf/manuals/372866b.pdf (accessed Dec. 20th, 2021).

[84] "NI PXIe-7857R Specifications - National Instruments." National Instruments. https://www.ni.com/pdf/manuals/378001a_02.pdf (accessed.

[85] "Intel Celeron Processor T3100 1M Cache 1.90 GHz 800 MHz FSB Product Specifications." Intel Corporation. https://ark.intel.com/content/www/us/en/ark/products/37258/intel-celeron-processor-t3100-1m-cache-1-90-ghz-800-mhz-fsb.html (accessed 30th Mar., 2022).

[86] "7 Series FPGAs Data Sheet: Overview (DS180)." Xilinx, Inc. https://www.xilinx.com/content/dam/xilinx/support/documents/data_sheets/ds180_7Series_Overview.pdf (accessed 30th Mar., 2022).

[87] "Single-Cycle Timed Loop FAQ for the LabVIEW FPGA Module - NI." National Instruments. https://knowledge.ni.com/KnowledgeArticleDetails?id=kA00Z000000P8sWSAS&l=en-GB (accessed 31st Mar., 2022).

[88] A. Rushton, *VHDL for Logic Synthesis*. New York, United Kingdom: John Wiley & Sons, Incorporated, 2011.

[89] M. Zuluaga. "Spiral Project: Sorting Network IP Generator." Spiral Project. http://spiral.net/hardware/sort/sort.html (accessed 7th Apr., 2022).

[90] H. S.-h. Chung, H. Wang, F. Blaabjerg, and M. Pecht, "Reliability of Power Electronic Converter Systems," ed: Institution of Engineering and Technology (The IET).

[91] P. Münch, S. Liu, and M. Dommaschk, "Modelling and Current Control of Modular Multilevel Converters Considering Actuator and Sensor Delays," in *2009 35th Annual Conference of IEEE Industrial Electronics*, 3-5 Nov. 2009 2009, pp. 1633-1638, doi: 10.1109/IECON.2009.5414756.

[92] T. Corrêa, F. Rodríguez, and E. Bueno, "Model-Based Latency Compensation for Network Controlled Modular Multilevel Converters," *MDPI Electronics*, vol. 8, no. 22, 2018.

[93] "Dataflow and the Enable Chain in FPGA VIs (FPGA Module) - LabVIEW 2018 FPGA Module Help - National Instruments." National Instruments. https://zone.ni.com/reference/en-XX/help/371599P-01/lvfpgaconcepts/fpga_sctl_and_enablechain/ (accessed 12th Apr., 2022).

[94] M. Zuluaga, P. Milder, and M. Püschel, "Computer Generation of Streaming Sorting Networks," in *DAC Design Automation Conference 2012*, 3-7 June 2012 2012, pp. 1241-1249, doi: 10.1145/2228360.2228588.

[95] T. Heath, P. R. Green, M. Barnes, and D. Kong, "Design, Construction and Testing of a Modular Multilevel Converter with a Distributed Control Architecture," in *15th IET International Conference on AC and DC Power Transmission (ACDC 2019)*, Coventry, UK, 5-7 Feb. 2019 2019: IET, pp. 1-6, doi: 10.1049/cp.2019.0073.

[96] "Model PVS60085MR, High Power Programmable DC Power Supplies - B&K Precision." B&K Precision. https://www.bkprecision.com/products/power-supplies/PVS60085MR-600v-8-5a-3kw-programmable-dc-power-supply.html (accessed Dec. 14th, 2021).

[97] "TA189 User's Guide." Pico Technology. https://www.picotech.com/download/manuals/ta189-30a-current-clamp-users-guide.pdf (accessed Dec. 15th, 2021).

[98] "BNC-2090A User Manual - National Instruments." National Instruments. https://www.ni.com/pdf/manuals/372101a.pdf (accessed Dec. 16th, 2021).

[99] S. Yang, Y. Tang, and P. Wang, "Distributed Control for a Modular Multilevel Converter," *IEEE Transactions on Power Electronics*, vol. 33, pp. 5578-5591, 2018, doi: 10.1109/TPEL.2017.2751254.

[100] *IEC 61850: Communication networks and systems for power utility automation*, 2021. [Online]. Available: https://webstore.iec.ch/publication/6028

[101] M. Adamiak, D. Baigent, and R. Mackiewicz, "IEC 61850 Communication Networks and Systems In Substations: An Overview for Users," Feb. 7th 2009. [Online]. Available: https://www.gegridsolutions.com/multilin/journals/issues/spring09/iec61850.pdf

[102] "NI PXIe-1071 User Manual." National Instruments. https://www.ni.com/pdf/manuals/373011d.pdf (accessed Dec. 3rd, 2021).

[103] "PXIe-6363 Specifications - National Instruments." National Instruments. https://www.ni.com/pdf/manuals/377776a.pdf (accessed Dec. 17th, 2021).

[104] "NI R Series Multifunction RIO Specifications - National Instruments." National Instruments. https://www.ni.com/pdf/manuals/372492c.pdf (accessed Dec. 20th, 2021).

[105] "Terasic - DE Boards - Cyclone - DE10-Nano Kit." Terasic. https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=165&No=1046&PartNo=1 (accessed Dec. 20th, 2021).

[106] "DAQ X Series: X Series User Manual." National Instruments. https://www.ni.com/pdf/manuals/370784k.pdf (accessed Dec. 3rd, 2021).

[107] "Digilent Digital Discovery Reference Manual." Digilent Inc. https://reference.digilentinc.com/reference/instrumentation/digital-discovery/start (accessed.

[108] J. Andrews, P. R. Green, and M. Barnes, "A PSCAD Processor-in-the-loop System for Hardware Evaluation of Power Converter Control Algorithms," in *The 10th International Conference on Power Electronics, Machines and Drives (PEMD 2020)*, 15-17 Dec. 2020 2020, vol. 2020, pp. 255-260, doi: 10.1049/icp.2021.1060.

[109] J. Carmona-Sanchez, P. R. Green, M. Barnes, and O. Marjanovic, "A Realistic Telecommunication Model for Electromagnetic Transient Simulations and Control Assessment of Multi-terminal VSC-HVDC Networks in PSCAD/EMTDC," in *15th IET International Conference on AC and DC Power Transmission (ACDC 2019)*, 5-7 Feb. 2019 2019, pp. 1-6, doi: 10.1049/cp.2019.0014.

[110] R. Marquardt, A. Lesnicar, and J. Hildinger, "Modulares Stromrichterkonzept für Netzkupplungsanwendung bei hohen Spannungen bei hohen Spannungen," presented at the ETG-Fachbericht, 2002.

APPENDICES

Appendix A - Pseudocode for Sorting Algorithms

For a given sorting algorithm, several implementations of the algorithm are usually possible whilst still maintaining the underlying algorithmic structure of the specific sorting algorithm. Pseudocode of the classical sorting algorithms implemented in this work is included here for reference purposes and to avoid any ambiguity around the structure of the algorithms. Pseudocode is used, rather than source code in a particular programming language, since it captures the basic structure of the algorithm without the added complexity of programming language syntax.

The pseudocode in the following listings assumes that array indexing is zero-based and that any variables used have been suitably declared before use. In each case, `A` is the array to be sorted, and a utility function: `length(A)` is provided which returns the number of elements in `A`.

A-1 Bubble Sort

```
function BubbleSort(A)

    n = length(A)

    while (n > 1)

        new_n = 0

        for (i = 1; i <= n-1; i++)

            if (A[i-1] > A[i]) then

                // do swap
                temp = A[i-1]
                A[i-1] = A[i]
                A[i] = temp
                new_n = i

            end if

        end for

        n = new_n

    end while

end function
```

Listing A-1: Bubble Sort algorithm pseudocode

A-2 Insertion Sort

```
function InsertionSort(A)

    i = 1

    while (i < length(A))

        x = A[i]
        j = i-1

        while (j >= 0) and (A[j] > x)

            A[j+1] = A[j]
            j = j-1

        end while

        A[j+1] = x
        i = i+1

    end while

end function
```

Listing A-2: Insertion Sort algorithm pseudocode

A-3 Merge Sort

The merge sort implementation used in this work is the top-down recursive form. The sorting algorithm is executed by calling the top level `MergeSort` function with the array bounds as the `lo` and `hi` arguments: `MergeSort(A, 0, length(A)-1)`

```
function MergeSort(A, lo, hi)

    if (lo > hi)
        return
    end if

    mid = (lo+hi) / 2

    MergeSort(A, lo, mid)
    MergeSort(A, mid+1, hi)
    merge(A, lo, mid, hi)

end function

function merge(A, lo, mid, hi)

    len_lo = mid-lo+1
    len_hi = hi-mid

    // copy data into temporary arrays
    for (i = 0; i < len_lo; i++)
        L[i] = A[lo+i]
    end for

    for (i = 0; i < len_hi; i++)
        H[i] = A[mid+i+1]
    end for

    i = 0
    j = 0
    k = 1

    while (i < len_lo) and (j < len_hi)

        if (L[i] <= H[j]) then

            A[k] = L[i]
            i++

        else

            A[k] = H[j]
            j++

            end if

            k++

        end while

        while (i < len_lo)

            A[k] = L[i]
            i++
            k++

        end while

        while (j < len_hi)

            A[k] = H[i]
            j++
            k++

        end while

    end function
```

Listing A-3: Merge Sort algorithm pseudocode

A-4 Quick Sort

The quick sort implementation used in this work is the top-down recursive form using the Hoare partition scheme [65]. Like Merge Sort, the sorting algorithm is executed by calling the top level `QuickSort` function with the array bounds as the `lo` and `hi` arguments: `QuickSort(A, 0, length(A)-1)`. The `floor()` function returns the greatest integer less than or equal to the value passed to the function.

```
function QuickSort(A, lo, hi)

    if (lo >= 0) and (hi >= 0) and (lo < hi) then

        // calculate the pivot element position
        // and partition the array
        p = partition(A, lo, hi)
        QuickSort(A, lo, p)
        QuickSort(A, p+1, hi)

    end if

end function

function partition(A, lo, hi)

    pivot = A[ floor((hi+lo) / 2) ]

    i = lo-1
    j = hi+1

    while true

        while (A[i] < pivot)

            i = i+1

        end while

        while (A[j] > pivot)

            j = j-1

        end while

        if (i >= j) then

            return j

        end if

```

```
            temp = A[i]
            A[i] = A[j]
            A[j] = temp

        end while

end function
```

Listing A-4: Quick Sort algorithm pseudocode

Appendix B - Source Code for Custom PSCAD/EMTDC Component

This appendix provides FORTRAN source code listings for the custom PSCAD/EMTDC capacitor balancing control (CBC) component used in the simulation model described in Chapter 5. The custom component was developed to facilitate the research into different CBC methods presented in Chapter 6. The source code listings are provided here, should the reader wish to inspect the implementation of the CBC methods and compare them with the flowchart depictions shown in Chapter 5.

The custom CBC component implements the required CBC methods (periodic, average tolerance band, and cell tolerance band) in a single component – that is – it is not necessary to replace the component on the PSCAD schematic canvas when changing CBC method. It makes extensive use of PSCAD script directives in the component definition script to control which code segments are compiled, based upon the CBC method selected. Some source code is common across all CBC methods.

To aid understanding, the source code listings divided into several sub-sections based upon whether the code is common across all CBC methods or specific to a particular method. Furthermore, some of the PSCAD definition script directives have been removed to prevent confusion with the source code itself. The sub-sections are outlined as follows:

- B-1: source code which is common across all CBC methods
- B-2: source code for evaluating the trigger conditions for the periodic CBC method
- B-3: source code for evaluating the trigger conditions for the average tolerance band CBC method
- B-4: source code for evaluating the trigger conditions for the cell tolerance band CBC method
- B-5: source code for the bubble sort algorithm subroutine, common across all CBC methods

The source code in Appendices B-2 to B-4 is substituted into the common source code in the location indicated in Appendix B-1 dependent upon the selected CBC method. The source code listings in this appendix form only part of the custom component; the PSCAD/EMTDC component can be downloaded from the repository in [26], should the reader wish to explore its operation further.

B-1 Common Source Code

```
! allocate space in storage array for previous timestep values
#STORAGE INTEGER:$(DimCBC)

! variable declarations
#LOCAL INTEGER i
#LOCAL INTEGER n
#LOCAL INTEGER j

#LOCAL INTEGER trigger_cbc
#LOCAL INTEGER my_nstori
#LOCAL INTEGER my_nstorf

#LOCAL INTEGER sort_swaps
#LOCAL INTEGER sort_comparisons

#LOCAL REAL vcap_curr_timestep $(DimCBC)
#LOCAL INTEGER idx_prev_timestep $(DimCBC)

#LOCAL REAL temp_vcap
#LOCAL INTEGER temp_idx

#LOCAL REAL vcap_sum
#LOCAL REAL vcap_avg

#LOCAL REAL atb_upper
#LOCAL REAL atb_lower
#ENDIF
#ENDIF

! initialise local storage array variables and increment
my_nstori = NSTORI
NSTORI = NSTORI + $DimCBC

! if t=0, initialise last timestep array with initial conditions
! else, retrieve previous time step values from storage arrays
IF (TIMEZERO .EQV. .TRUE.) THEN

    DO i=1,$DimCBC
        idx_prev_timestep(i) = i
    ENDDO

ELSE

    DO i=1,$DimCBC
        idx_prev_timestep(i) = STORI(my_nstori+i-1)
    ENDDO

ENDIF

```

```
77  ! reset sorting algorithm swaps/comparisons variables
78  sort_swaps = 0
79  sort_comparisons = 0
80
81  ! reset trigger
82  trigger_cbc = 0
83
84  ! clear triggered output port
85  $TrigOut = 0
86
87  ! [...] continued below
```

Listing B-1: Custom component common FORTRAN source code, part 1

The CBC method-specific code which determines whether capacitor balancing should be triggered is inserted after Line 59 in Listing B-1:. The source code which follows the CBC method-specific code is shown below in Listing B-2.

```
! [...]
! initialise arrays
DO i=1,$DimCBC

    $Idx(i) = i
    vcap_curr_timestep(i) = $Vcap(i)

ENDDO

! if CBC has been triggered, execute balancing
IF (trigger_cbc .EQ. 1) THEN

    ! set triggered output port
    $TrigOut = 1

    ! do CBC sort
    ! sort based on arm current, always placing the submodules to
    ! be inserted in the lowest (ie. 1, 2...) array indices

    ! if arm current is positive, sort ascending
    ! (so that SMs with lowest CVs end up in lowest indices)
    IF ($Iarm .GE. 0) THEN

        ! bubble sort ascending
        CALL bubble_sort_ascending($DimCBC, vcap_curr_timestep,
          $Idx, sort_swaps, sort_comparisons)

    ! else if arm current is negative, sort descending
    ! (so that SMs with highest CVs end up in lowest indices)
    ELSE

        ! bubble sort descending
        CALL bubble_sort_descending($DimCBC, vcap_curr_timestep,
          $Idx, sort_swaps, sort_comparisons)

    ENDIF

ELSE

    ! CBC has not been triggered, so write previous
    ! timestep index values to outputs
    DO i=1,$DimCBC
        $Idx(i) = idx_prev_timestep(i)
    ENDDO

ENDIF

! update index storage array
DO i=1,$DimCBC
    STORI(my_nstori+i-1) = $Idx(i)
ENDDO

! write sorting algorithm statistics to component output ports
$SwapOut = sort_swaps
$CompOut = sort_comparisons
```

Listing B-2: Custom component common FORTRAN source code, part 2

B-2 Periodic CBC Method Source Code

```fortran
! CBC Mode: Periodic (External Trigger)
! if external trigger input port == 1, then run balancing
IF ($TrigIn .EQ. 1) THEN
    trigger_cbc = 1
ENDIF
```

Listing B-3: Periodic CBC method FORTRAN source code

B-3 Average Tolerance Band CBC Method Source Code

```fortran
! CBC Mode: Average Tolerance Band (ATB)
! calculate average capacitor voltage
vcap_sum = 0

DO i=1,$DimCBC
    vcap_sum = vcap_sum + $Vcap(i)
ENDDO

vcap_avg = vcap_sum / $DimCBC

! for each SM check that Vcap is within the tolerance band
! if it is not, trigger the CBC loop

! tolerance band is given in kV
DO i=1,$DimCBC

    IF (ABS($Vcap(i)-vcap_avg) .GT. $ATBToleranceBand) THEN
        trigger_cbc = 1
    ENDIF

ENDDO
```

Listing B-4: Average Tolerance Band CBC method FORTRAN source code

B-4 Cell Tolerance Band CBC Method Source Code

```fortran
! CBC Mode: Cell Tolerance Band (CTB)
! for each SM, check that Vcap is within the max/min limits
DO i=1,$DimCBC

    IF (($Vcap(i) .GT. $CTBMaxVcap) .OR. ($Vcap(i) .LT.
      $CTBMinVcap)) THEN
        trigger_cbc = 1
    ENDIF

ENDDO
```

Listing B-5: Cell Tolerance Band CBC method FORTRAN source code

B-5 Bubble Sort Subroutine Source Code

When capacitor balancing is triggered, the submodule capacitor voltages are sorted using the bubble sort algorithm. The direction of the sort (ascending or descending) is dictated by the direction of the arm current. Listing B-6 and Listing B-7 show the source code for the bubble sort implementation used in the custom component for ascending and descending sorting direction respectively. The bubble sort algorithm is implemented as a FORTRAN subroutine which is called from the custom component.

```fortran
subroutine bubble_sort_ascending(n, vcap_array, idx_array, swaps,
  comparisons)

    ! dummy arguments
    integer n, idx_array(*), swaps, comparisons
    real vcap_array(*)

    integer i, j, temp_idx
    real temp_vcap

    ! bubble sort algorithm
    do i=n-1,1,-1

        do j=1,i

            if (vcap_array(j+1) .LT. vcap_array(j)) then

                temp_vcap = vcap_array(j+1)
                vcap_array(j+1) = vcap_array(j)
                vcap_array(j) = temp_vcap

                temp_idx = idx_array(j+1)
                idx_array(j+1) = idx_array(j)
                idx_array(j) = temp_idx

                swaps = swaps + 1

            else

                comparisons = comparisons + 1

            endif

        enddo

    enddo

end subroutine
```

Listing B-6: Bubble Sort algorithm FORTRAN source code, ascending sort direction

```fortran
subroutine bubble_sort_descending(n, vcap_array, idx_array,
  swaps, comparisons)

    ! dummy arguments
    integer n, idx_array(*), swaps, comparisons
    real vcap_array(*)

    integer i, j, temp_idx
    real temp_vcap

    ! bubble sort algorithm
    do i=n-1,1,-1

        do j=1,i

            if (vcap_array(j+1) .GT. vcap_array(j)) then

                temp_vcap = vcap_array(j+1)
                vcap_array(j+1) = vcap_array(j)
                vcap_array(j) = temp_vcap

                temp_idx = idx_array(j+1)
                idx_array(j+1) = idx_array(j)
                idx_array(j) = temp_idx

                swaps = swaps + 1

            else

                comparisons = comparisons + 1

            endif

        enddo

    enddo

end subroutine
```

Listing B-7: Bubble Sort algorithm FORTRAN source code, descending sort direction

Appendix C - Selection of Maximum Allowable Control Cycle Delay Threshold

This appendix provides a more detailed analysis of the selection of the value of $M_{control-threshold}$ used in Section 6.4 of Chapter 6 in this thesis. The motivations for applying a threshold to the number of missed control cycles due to the sorting algorithm execution delay are outlined first, followed by an overview of submodule (SM) capacitor voltage dynamics. Simulation results from the PSCAD/EMTDC model of the 351-level half-bridge SM (HB-SM) modular multilevel converter (MMC) are then presented to explain the selection of $M_{control-threshold}$ = 8 in Section 6.4. Several caveats to this approach are then identified.

Motivation for Selection of $M_{control-threshold}$

As outlined in Section 6.4.2, a delay within the CBC will lead to degraded capacitor balancing performance and a larger deviation in SM capacitor voltage from $V_{cap-nom}$. This was shown originally by the results presented in [92], where the maximum deviation of the SM capacitor voltage was measured for an increasing number of samples delay within the CBC loop. Furthermore, it was also shown in [92] that the CBC loop is able to tolerate a certain amount of delay in the loop, whilst still maintaining SM capacitor voltages within the desired range. The delay in the loop may be caused by the communication network between the controller and the SM, or by a processing delay in the CBC loop. In this thesis, the delay being investigated is due to the sorting algorithm execution time.

Motivated by the work presented in [92], it was decided to simplify the analysis of the sorting algorithm execution time data in Section 6.4.3 by applying a threshold value of the number of control cycles missed due to the sorting algorithm execution delay. The threshold value was termed $M_{control-threshold}$. Only the number of instances where the instantaneous timestep value of $M_{control}$ (termed $M_{control}[k]$, caused by the sorting algorithm execution delay) exceeded $M_{control-threshold}$ were counted. This approach was taken since, in instances where $M_{control}[k] < M_{control-threshold}$, the effect of the execution delay upon SM capacitor voltage balancing could be assumed to be negligible.

Submodule Capacitor Voltage Ripple

Submodule capacitor voltages naturally exhibit a ripple during converter operation due to the charging and discharging of the capacitor by the arm current when the SM is in the inserted state. The maximum allowable range for the SM capacitor voltage ripple is typically dictated by several factors, including:

- Capacitor working voltage range
- Capacitor maximum ripple current
- Submodule insulated gate bipolar transistor (IGBT) maximum blocking voltage
- Maximum voltage ratings of other SM components (eg. bypass switch, energy harvesting circuit for the SM auxiliary circuitry, etc.)

In addition to ensuring that SM component ratings are not exceeded, it is important that the SM capacitor voltages are balanced at or near to the nominal value, $V_{cap-nom}$, so that each output voltage step is equal to V_{dc}/N_{SM}. This ensures that the arm output voltage waveform is able to accurately track the input reference from the arm voltage controller. The capacitor balancing control (CBC) loop works to ensure that all of these control objectives are met.

Typically, SM capacitors are sized so that capacitor voltage ripple is within ± 5 % of $V_{cap-nom}$ [28, 31]. The required SM capacitance can be calculated analytically using the approach presented in [110], or alternatively using the estimate of 30-40 kJ of stored energy per MVA proposed in [28].

Sum Capacitor Voltage Ripple

As explained in Section 2.1.1, the instantaneous sum of the SM capacitor voltages in a single arm ($V_{capu,l}^{\Sigma}(t)$) exhibits a ripple due to summing of individual SM capacitor voltages which also exhibit a ripple. The maximum range of $V_{capu,l}^{\Sigma}(t)$ will typically occur when the converter is operating at maximum power or possibly during some transients [31], since in these scenarios the range of the arm current in each phase is also at its maximum. This causes the capacitors of SMs in the inserted state to charge or discharge faster. Therefore, it can be concluded that operating at maximum rated power represents a worst-case scenario for the CBC loop, since SM capacitor voltages will deviate from $V_{cap-nom}$ by a larger amount in a given time window. Based on this analysis, the model was configured to import (DC side to AC side power flow, inverter mode) the maximum rated active and reactive power of 1200 MW and 400 MVAr for the selection of $M_{control-threshold}$.

The range and profile of the sum SM capacitor voltage ripple ($V_{capu,l}^{\Sigma}(t)$) is dictated by the converter operating point and SM capacitance – that is – it is not dependent upon the chosen CBC method or whether or not individual SMs are balanced. As a result, $V_{capu,l}^{\Sigma}(t)$ cannot be used to measure and compare the performance of different CBC methods, or to measure how a given CBC method is affected by a delay in the loop.

Maximum Capacitor Voltage Deviation

Instead, a method of measuring the deviation of individual SM capacitor voltages is required. The authors in [92] proposed a measured described as “maximum deviation to the average capacitor voltage”, however this was poorly defined, despite the fact that it showed that increasing the delay in the CBC loop increases the maximum deviation. Therefore, a method to measure the deviation of individual SM capacitor voltages which can be used to assess the performance of CBC is defined here.

As described in Section 2.1.1, the nominal SM capacitor voltage, $V_{cap-nom}$, is calculated using the nominal DC bus voltage, V_{dc}, and the number of SMs per arm, N_{SM}, according to Equation C-1, first introduced in Section 2.1.1 but replicated here for ease.

$$V_{cap-nom} = \frac{V_{dc}}{N_{SM}} \tag{C-1}$$

In the 351-level HB-SM MMC model used in this work, V_{dc} = 640 kV and N_{SM} = 350, therefore $V_{cap-nom}$ = 1.83 kV to 3 s.f. It is important to remember that only the long-term mean SM capacitor voltage will actually converge to this value, since the instantaneous value of SM capacitor voltage exhibits a ripple as explained earlier.

A further quantity, $\Delta V_{cap}^{i}[k]$, can be defined according to Equation C-2. This is termed the instantaneous capacitor voltage deviation from the nominal and is calculated per-SM (index i) in each sampling instant (or simulation timestep), k.

$$\Delta V_{cap}^{i}[k] = V_{cap}^{i}[k] - V_{cap-nom} \tag{C-2}$$

Since all SM capacitor voltages will exhibit a deviation from the nominal, this measure alone cannot be used to compare the balancing performance of different CBC methods with or without a delay in the loop. Instead, the maximum instantaneous deviation from the nominal can be used, $\Delta V_{cap-max}[k]$, calculated according to Equation C-3.

$$\Delta V_{cap-max}[k] = \max{}_{i=1}^{N_{SM}}\left(\left|\Delta V_{cap}^{i}[k]\right|\right) \tag{C-3}$$

The max() operator searches the list of SM capacitor voltages (i = 1..N_{SM}) and finds the SM with the largest absolute deviation from the nominal. Using this measure, it is possible to obtain an envelope within which all other SM capacitor voltages lie. A larger envelope corresponds to worse capacitor voltage balancing between SMs. The maximum instantaneous deviation from the nominal can also be expressed as a percentage of $V_{cap-nom}$ using Equation C-4. This allows direct comparison with the typical target SM capacitor voltage ripple of ± 5 % specified earlier.

$$\%\Delta V_{cap-max}[k] = \frac{\Delta V_{cap-max}[k]}{V_{cap-nom}} \times 100 \tag{C-4}$$

Simulation Results

To obtain a value for $M_{control-threshold}$, the PSCAD/EMTDC model of the 351-level HB-SM MMC was used to generate capacitor voltage data for three CBC methods:

1. Periodic CBC operating at f_{s-CBC} = 1 kHz
2. Average tolerance band (ATB) with a tolerance band of ± 5 % around the instantaneous mean SM capacitor voltage, $\overline{V_{cap}}[k]$
3. Cell tolerance band (CTB) with a tolerance band of ± 0.091 kV around the nominal SM capacitor voltage, $V_{cap-nom}$

Each CBC method was tested with a delay in the loop of 0, 2, 4, 8, 16 control cycles (corresponding to a delay of 0 µs, 100 µs, 200 µs, 400 µs, and 800 µs with a 50 µs simulation timestep) by inserting an e^{-sT} delay between the CBC component output and the firing signal generator block, as shown in Figure C-1. This effectively introduced a delay in SM switching for capacitor balancing purposes.

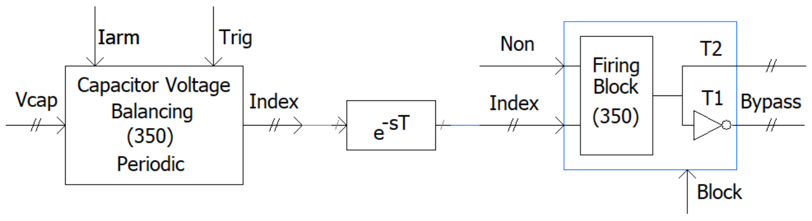


Figure C-1: PSCAD/EMTDC MMC model e^{-sT} delay block location

As stated earlier, the converter was configured in inverter mode to import the maximum rated active and reactive power of 1200 MW (P^*) and 400 MVAr (Q^*). The converter level response is shown in Figure C-2.

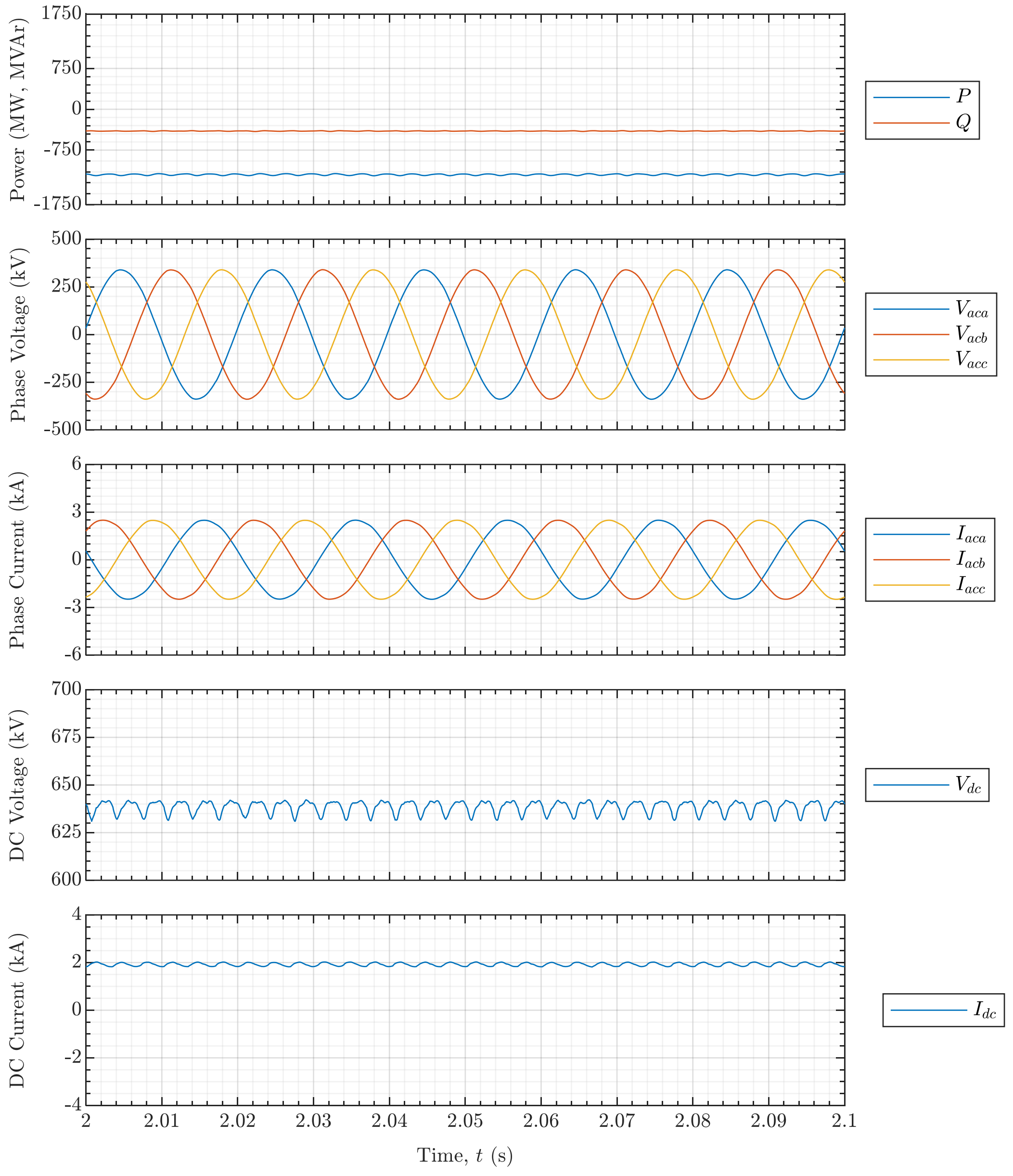
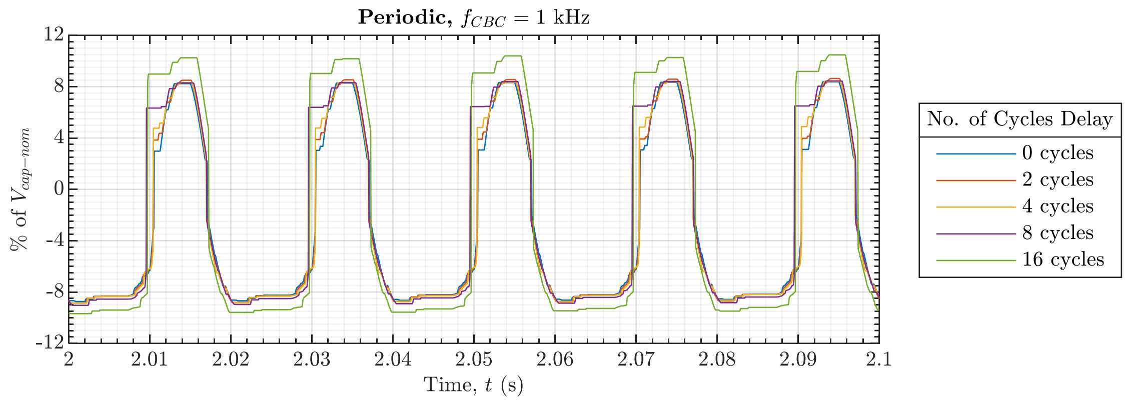
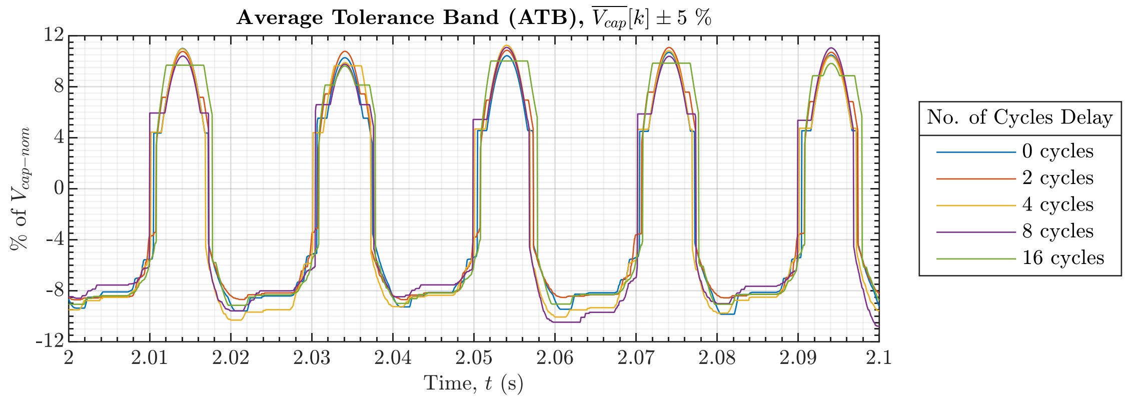


Figure C-2: Converter level response, steady state operation at $P^* = -1200$ MW, $Q^* = -400$ MVAr (import, inverter mode)

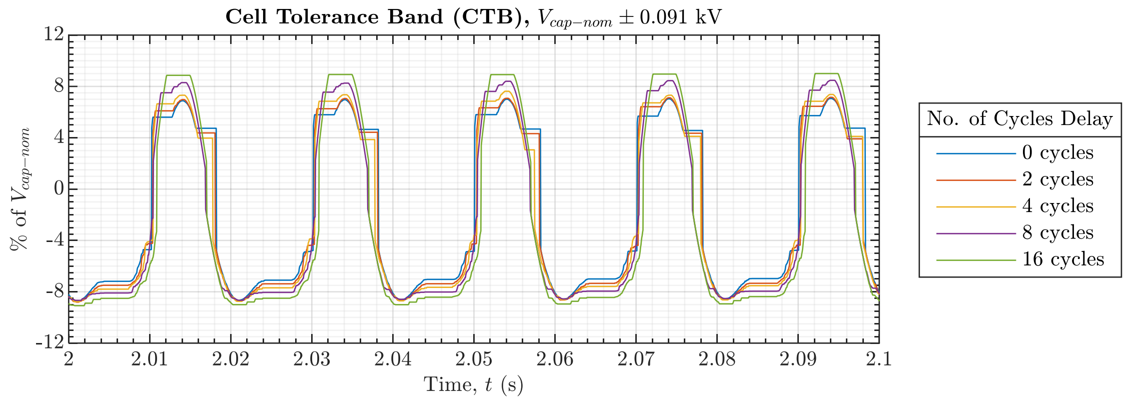
For each CBC method and delay value, the phase A upper arm SM capacitor voltages were logged to a file and $\%\Delta V_{cap-max}[k]$ was calculated at each timestep between t = 2.0 seconds to t = 2.1 seconds where the converter had reached steady state. This data is plotted in Figure C-3(a) to (c).



(a)



(b)



(c)

Figure C-3: Maximum instantaneous capacitor voltage deviation from $V_{cap-nom}$ as a percentage of $V_{cap-nom}$ at different delay lengths for (a) periodic, (b) average tolerance band, and (c) cell tolerance band CBC methods

It is important to note that the maximum deviation as a percentage of $V_{cap-nom}$ is greater than the ideal value of ± 5 % even where the delay is 0 cycles (no delay). This is due to under-sizing of the SM capacitance in the model caused by using the lower value of 30 kJ/MVA to calculate the stored energy requirement [80].

It can be seen from Figure C-3(a) and (c) that as the number of cycles delay is increased, the maximum deviation from $V_{cap-nom}$ also increases. Across all CBC methods and number of cycles delay, the maximum deviation occurs at a point corresponding to the zero crossing of the arm current and the point on the SM insertion index wave (N_{onua}) where nearly all SMs are inserted. This is shown in the zoomed-in plot in Figure C-4 for the periodic CBC case at $t \approx 2.034$ seconds.

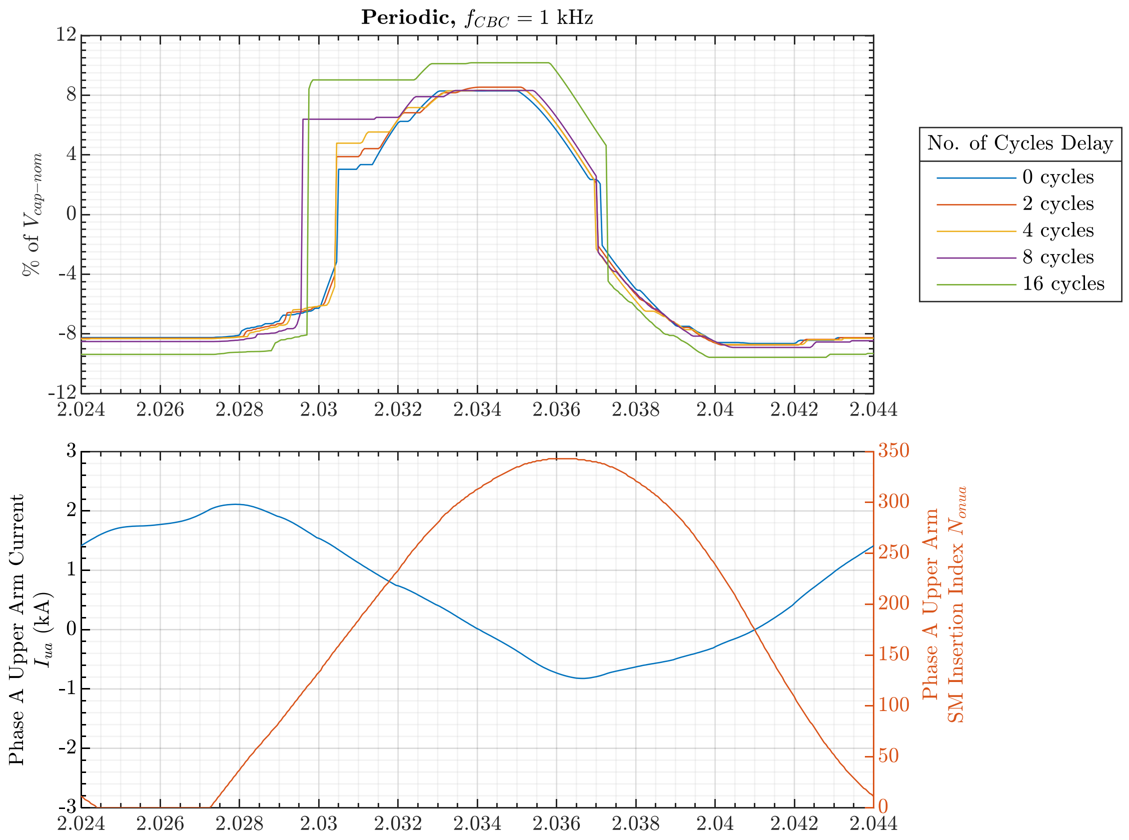


Figure C-4: Zoomed-in plot of maximum instantaneous deviation from $V_{cap-nom}$ as a percentage of $V_{cap-nom}$ at different delay lengths (top), and phase A upper arm current (bottom, left-hand y-axis) and phase A upper arm SM insertion index (bottom, right-hand y-axis)

Returning to Figure C-3(a) and (c), the increase in maximum deviation as the delay is increased is particularly pronounced above 8 cycles in both cases, focussing upon the regions where the curves are furthest apart. For periodic CBC, this occurs on the rising edge of the deviation curves for 8 and 16 cycles delay, and at the top of the curve for the 16 cycles delay case. In the case of the CTB method, the biggest difference between the curves occurs on the positive peak and in the plateau region on the negative peak. As a result, 8 cycles was chosen as the value for $M_{control-threshold}$.

Focussing upon Figure C-3(b) for the ATB method, the same relationship between increasing maximum deviation and increasing delay is not observed. This may be due to the fundamental operating principle of the ATB method, the choice of the tolerance band width, or other factors. Further research into the causes of this behaviour was outside the scope and timescale of this project, however the analysis methods presented in this appendix can be used to form the basis of this investigation.

Furthermore, it is important to note that the number of cycles delay required to cause a significant maximum deviation is highly dependent upon the chosen CBC method and converter set point. The analysis presented here aims to represent the worst-case scenario, however, other converter operating points may cause a non-negligible maximum deviation with a smaller number of cycles delay.

Appendix D - Supplementary Plots for Chapter 6

This appendix contains supplementary plots for the comparison of capacitor balancing control (CBC) method execution time window and sorting algorithm execution time carried out in Section 6.4 of Chapter 6.

Plots are grouped according to the three scenarios simulated in Section 6.4.3. The first group of plots in each sub-section show, for each CBC method (periodic, average tolerance band, and cell tolerance band):

- Phase A upper arm current (I_{ua})
- Phase A upper arm SM insertion index (N_{onua})
- Phase A upper arm SM capacitor voltages ($V_{capua}^{i=1..350}$)
- CBC loop sampling instances as a density plot

The density plot displays each CBC loop sampling instant as a thin grey line. The spacing between lines is equal to the CBC loop sampling period, T_{s-CBC}. These plots can be used to visualise the points on the arm current, SM insertion index, or SM capacitor voltage plots where the CBC loop is triggered at a higher or lower frequency.

The second group of plots in each scenario show the number of control cycles missed by the sorting algorithm ($M_{control}$) due to the sorting algorithm execution time in histogram form. When compared to the thresholding approach used in Chapter 6, the histograms provide a more detailed insight into the distribution of $M_{control}$ for each sorting algorithm implementation and CBC method.

D-1 Scenario 1: Steady State Power Import

Periodic

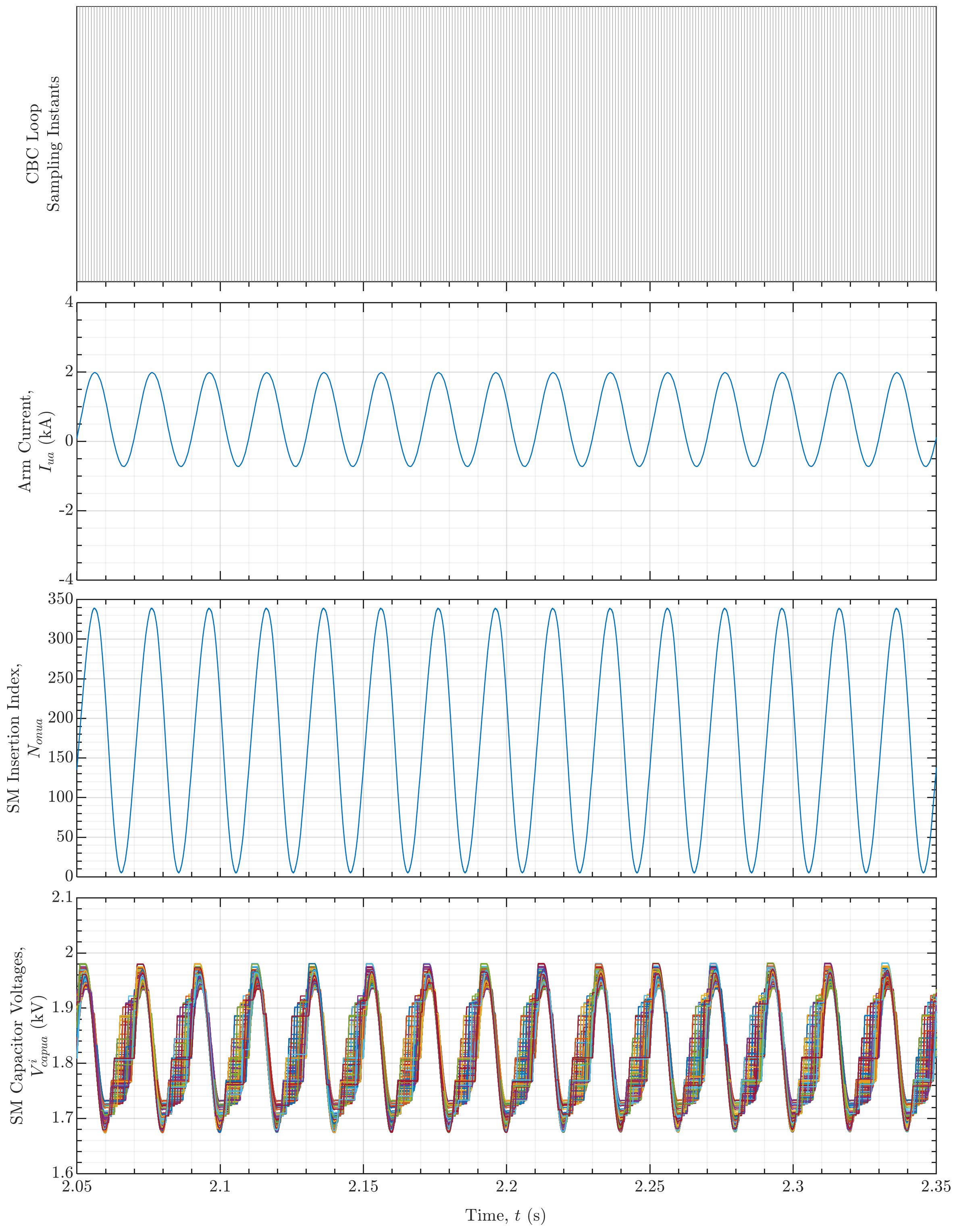


Figure D-5: Upper arm electrical quantities, scenario 1, periodic CBC

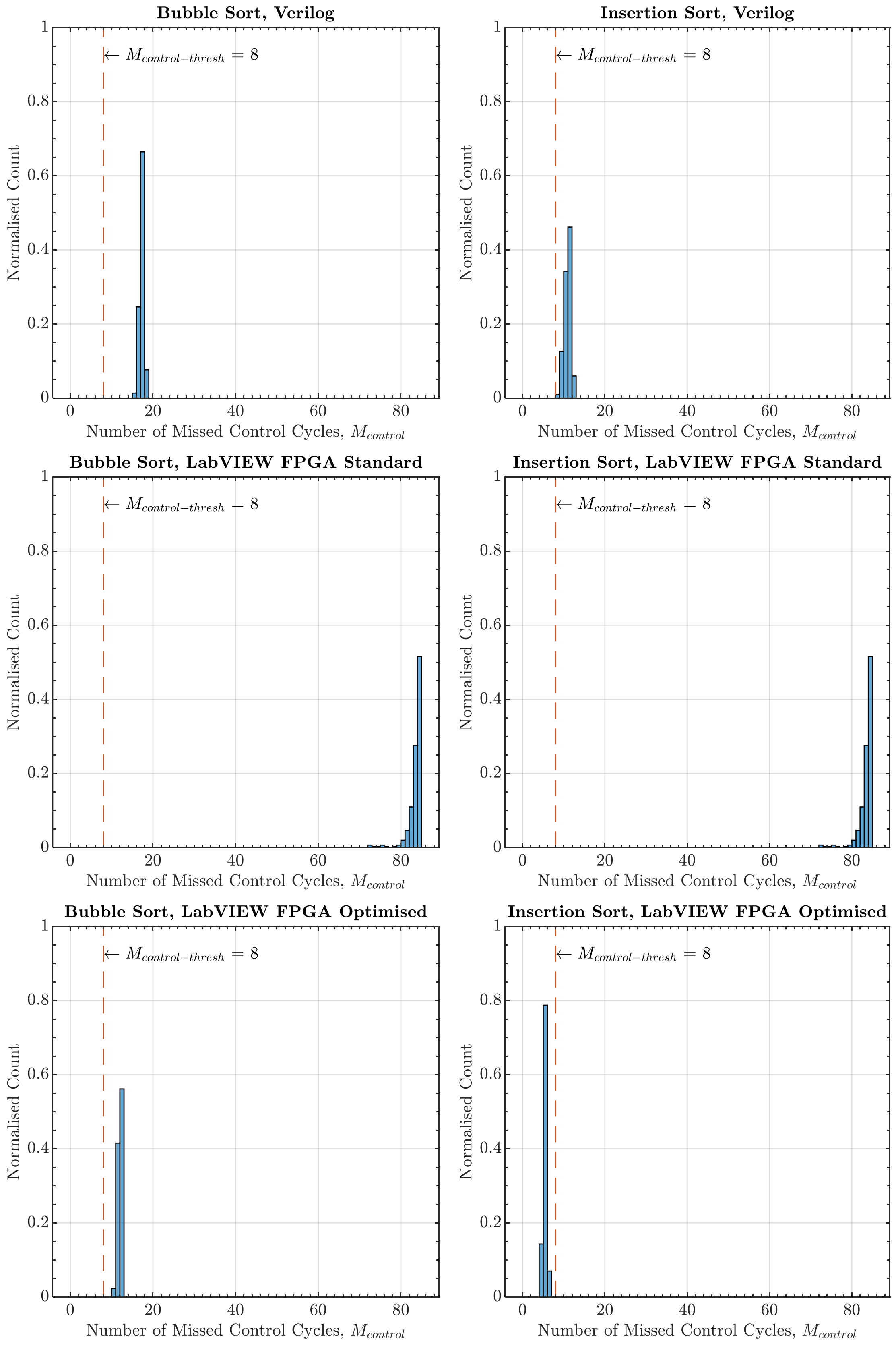


Figure D-6: Sorting algorithm missed control cycle histograms, scenario 1, periodic CBC

Average Tolerance Band

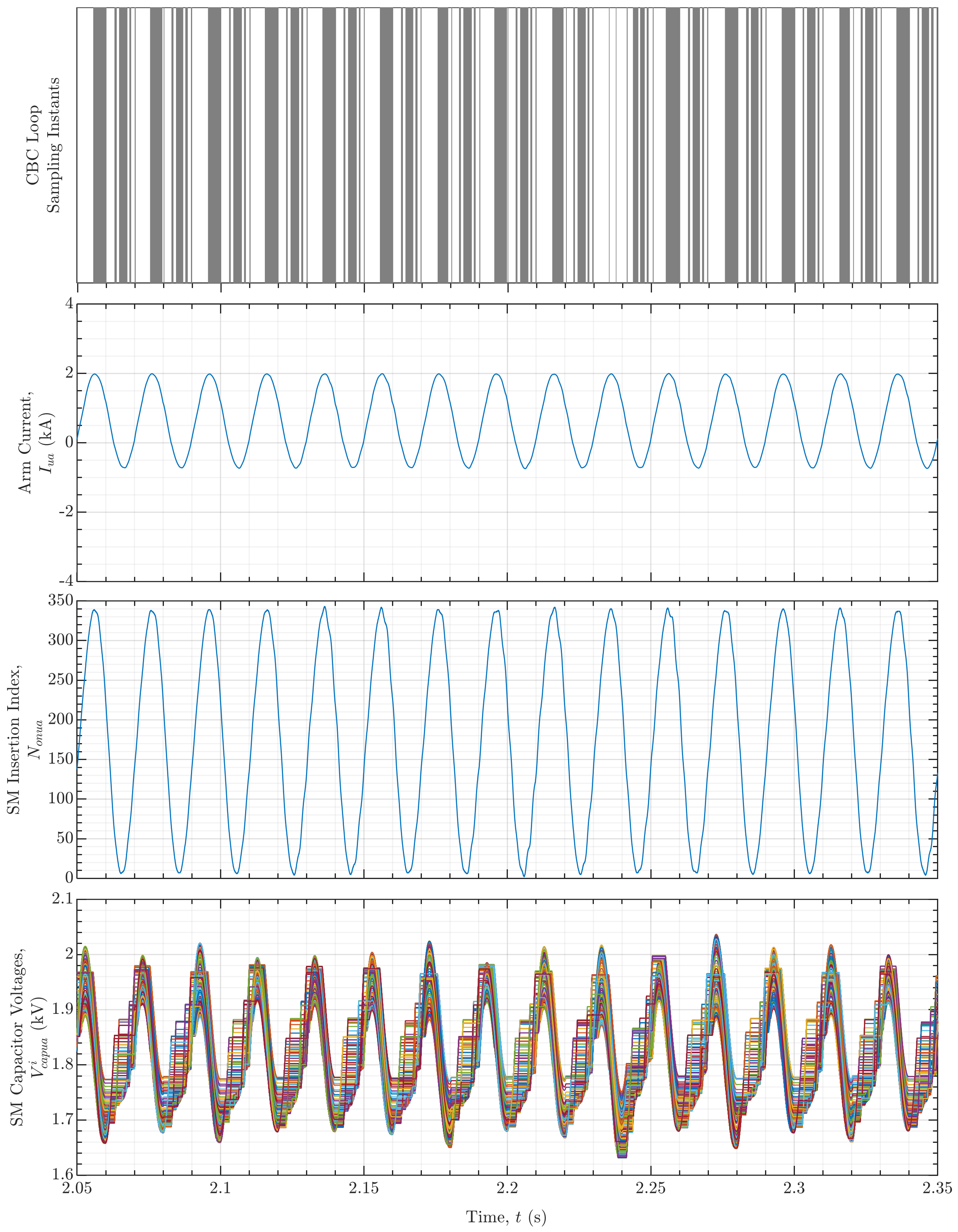


Figure D-7: Upper arm electrical quantities, scenario 1, average tolerance band CBC

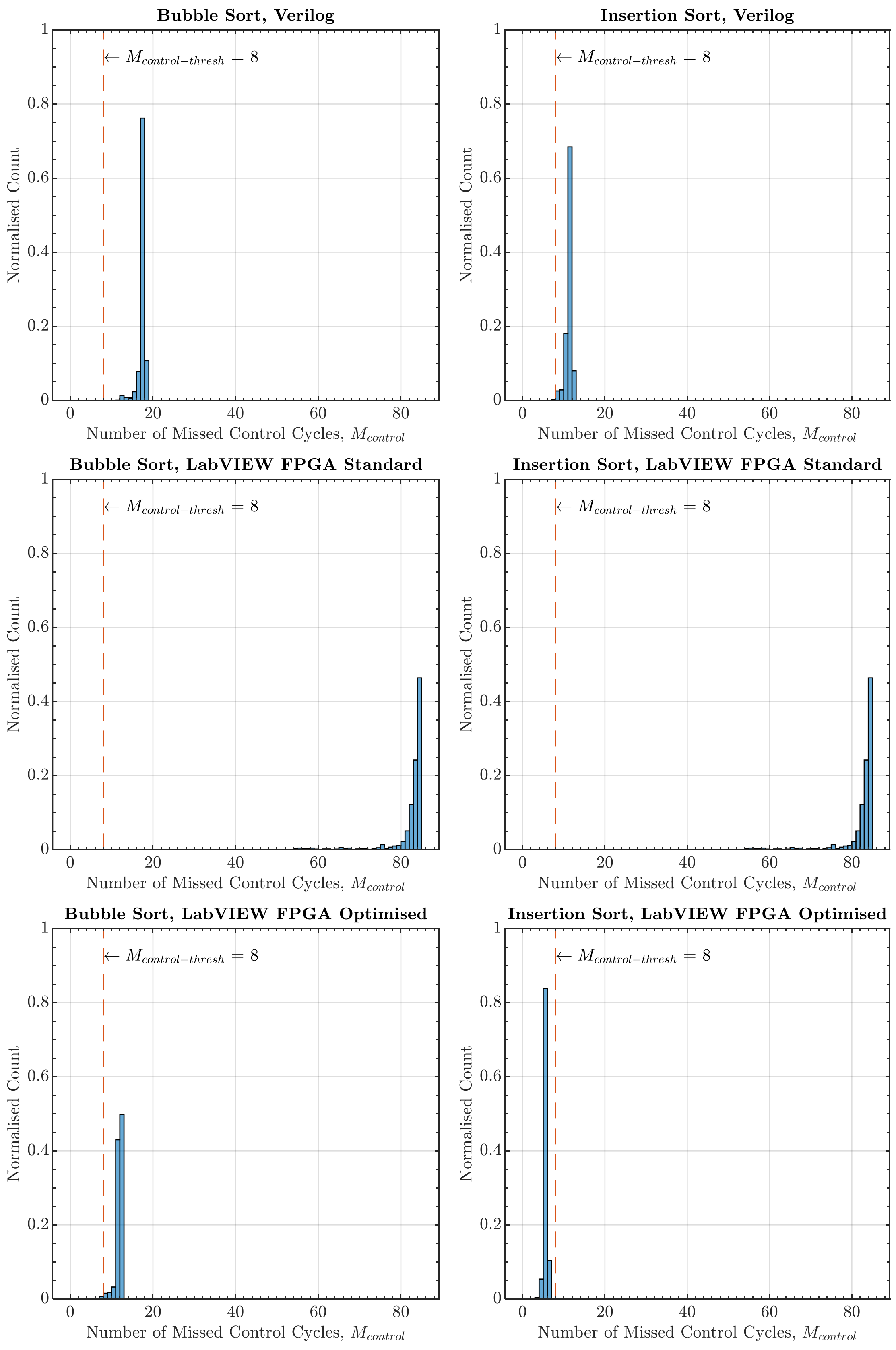


Figure D-8: Sorting algorithm missed control cycle histograms, scenario 1, average tolerance band CBC

Cell Tolerance Band

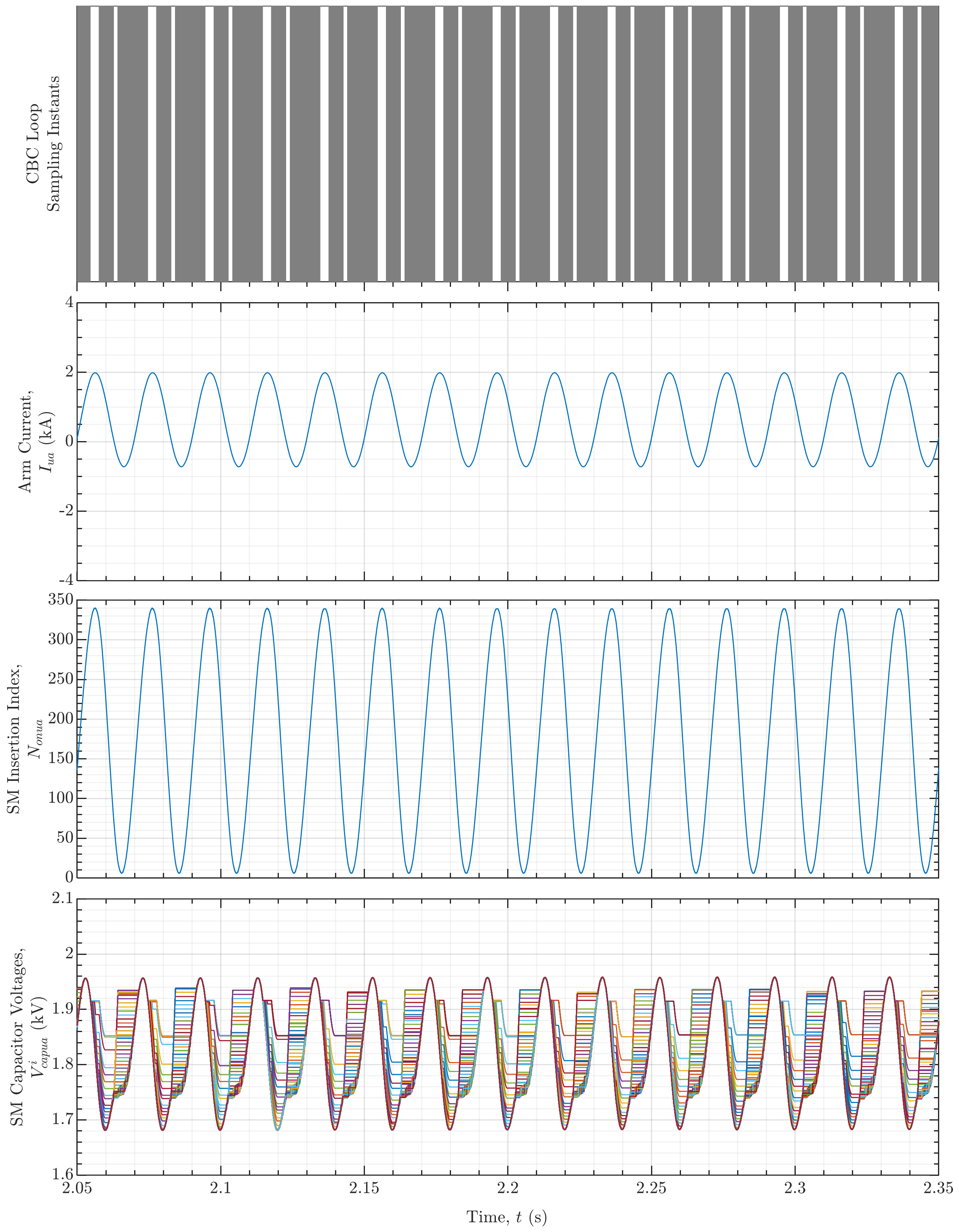


Figure D-9: Upper arm electrical quantities, scenario 1, cell tolerance band CBC

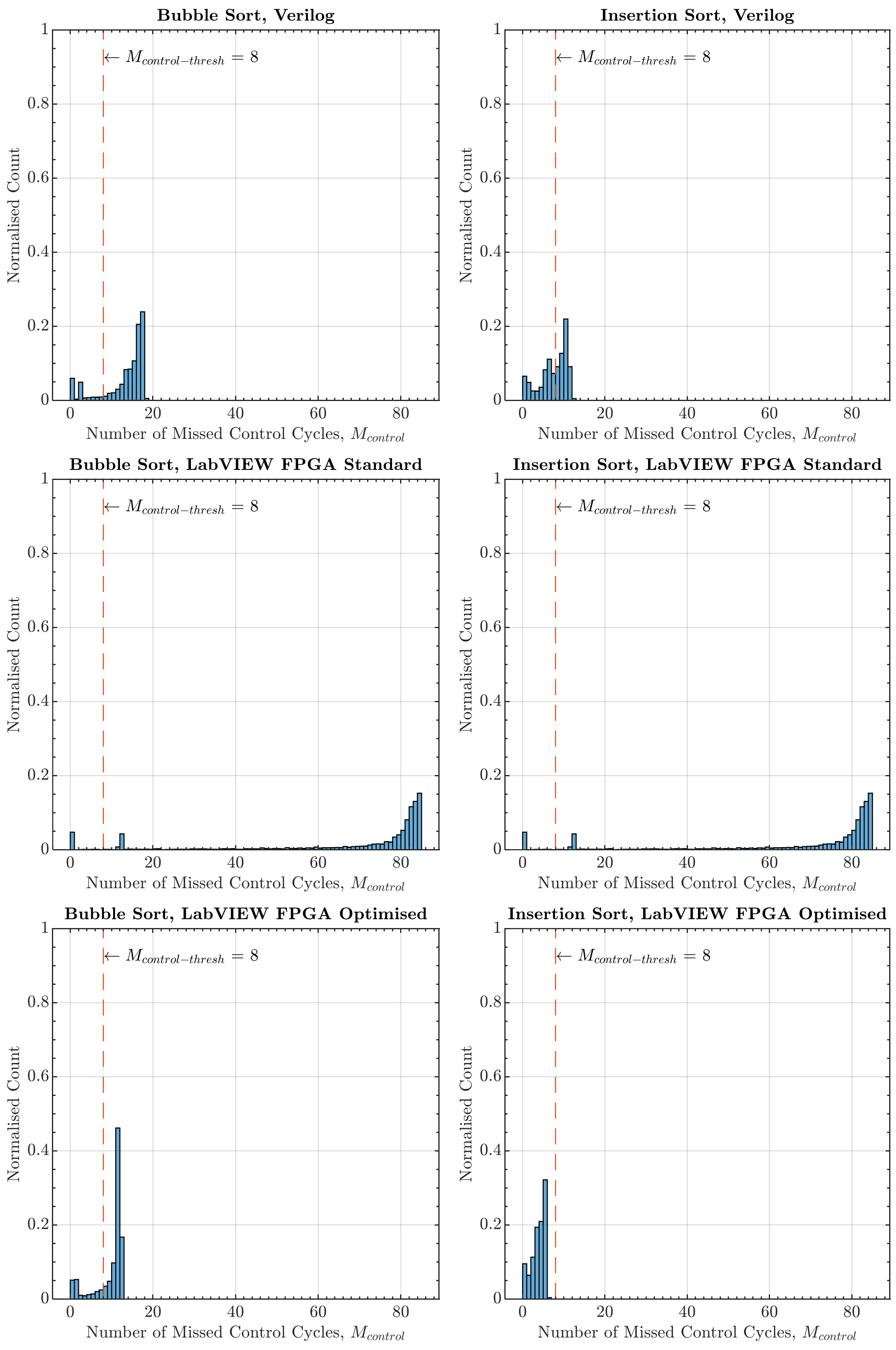


Figure D-10: Sorting algorithm missed control cycle histograms, scenario 1, cell tolerance band CBC

D-2 Scenario 2: Power Flow Reversal

Periodic

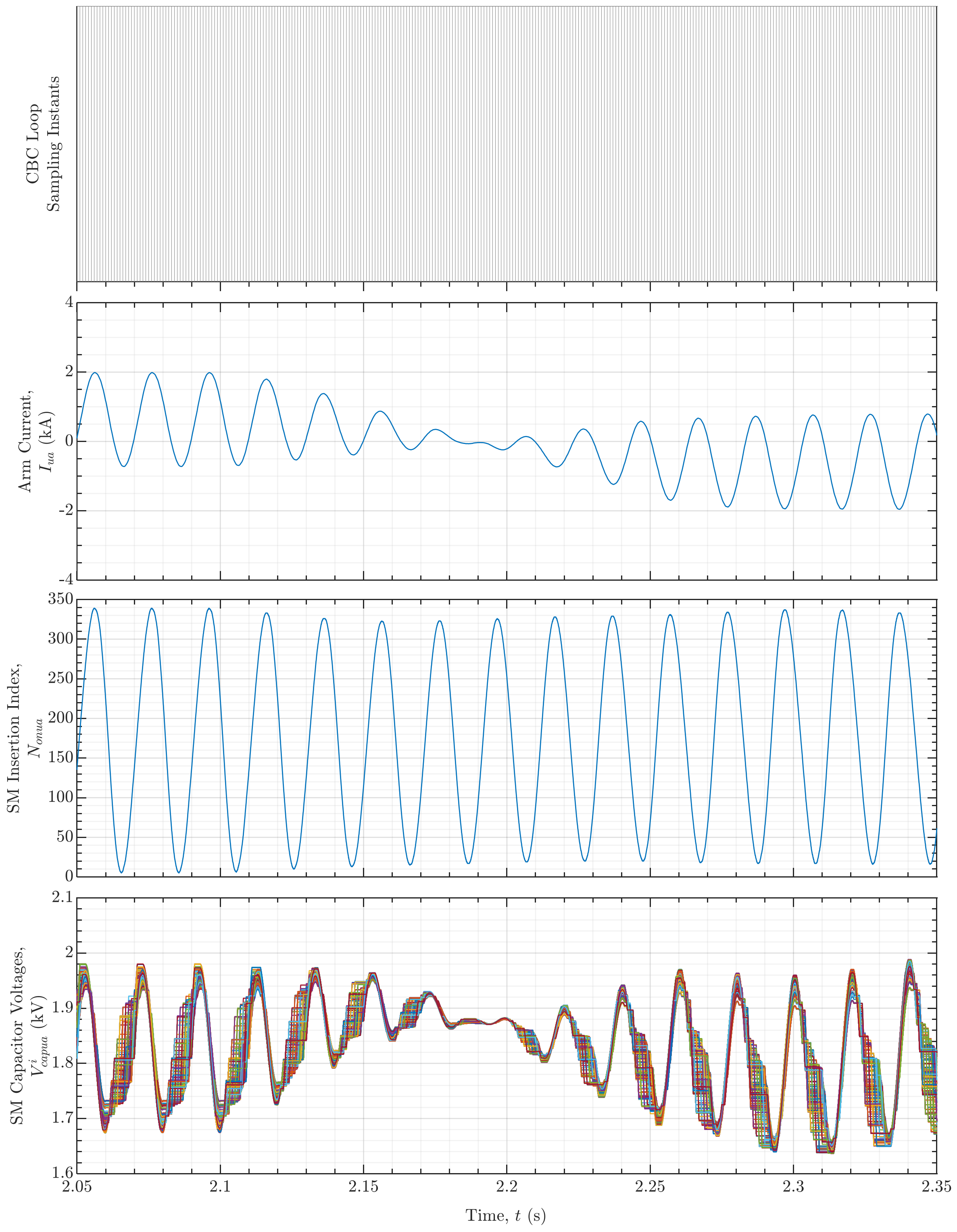


Figure D-11: Upper arm electrical quantities, scenario 2, periodic CBC

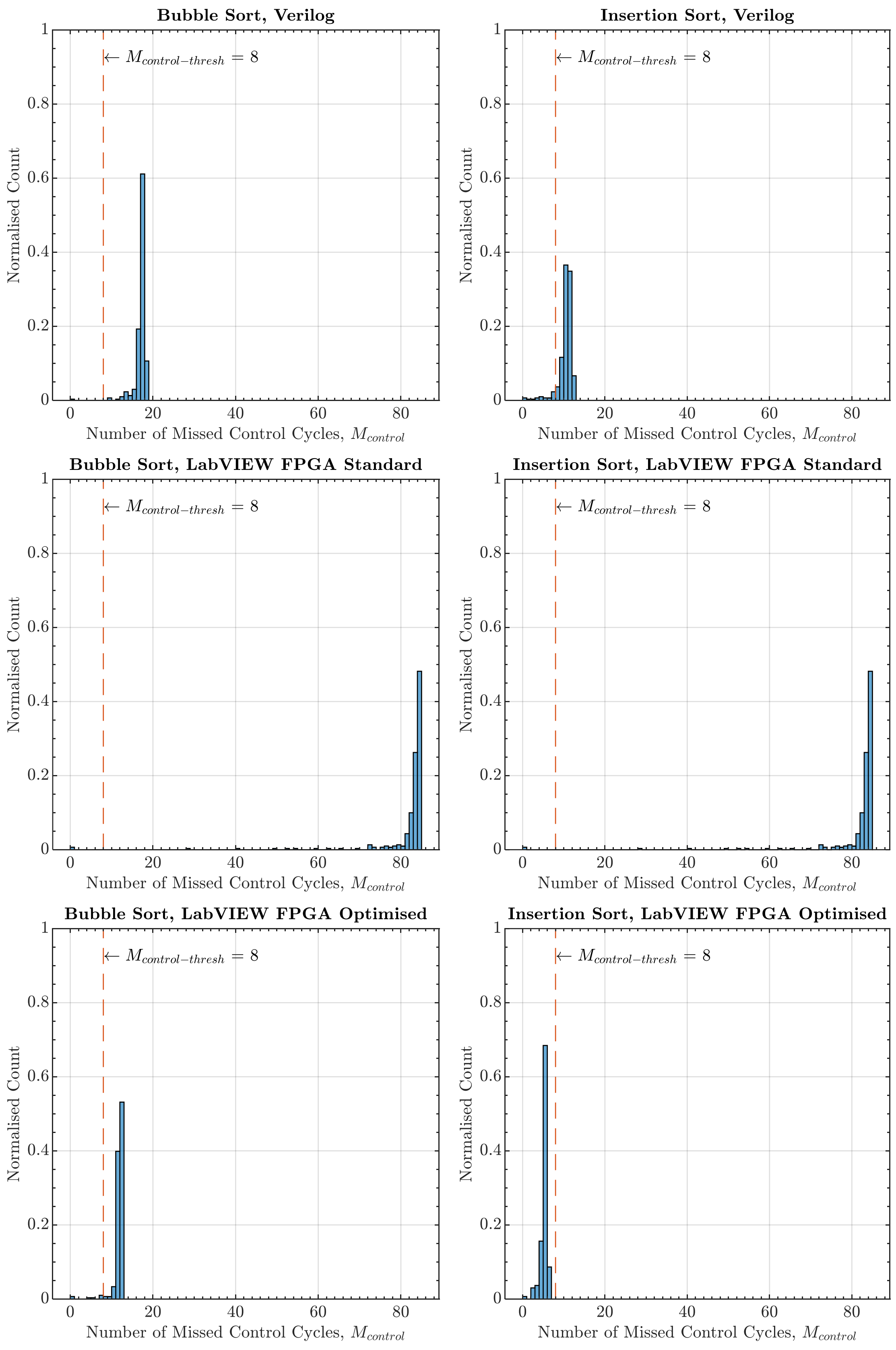


Figure D-12: Sorting algorithm missed control cycle histograms, scenario 2, periodic CBC

Average Tolerance Band

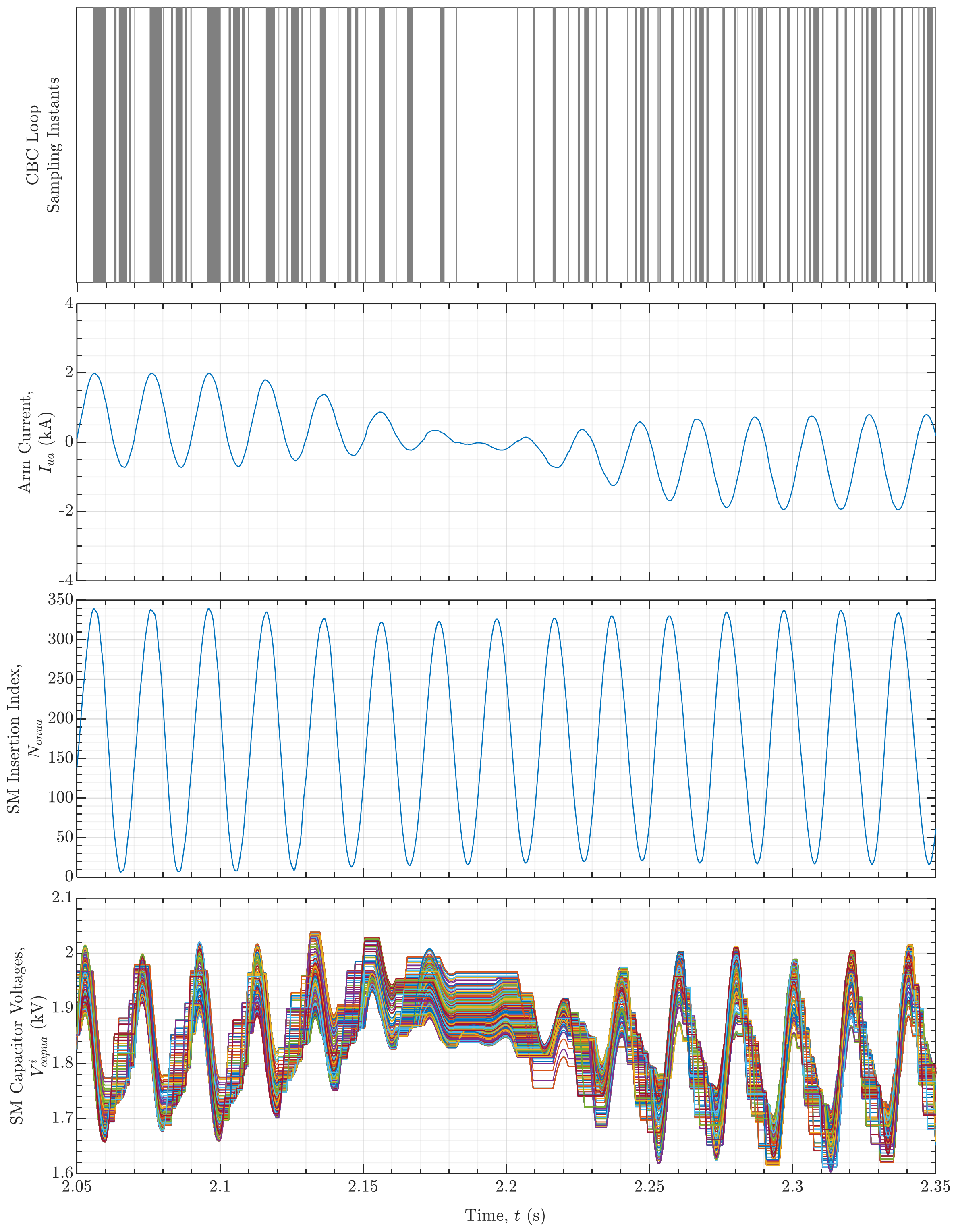


Figure D-13: Upper arm electrical quantities, scenario 2, average tolerance band CBC

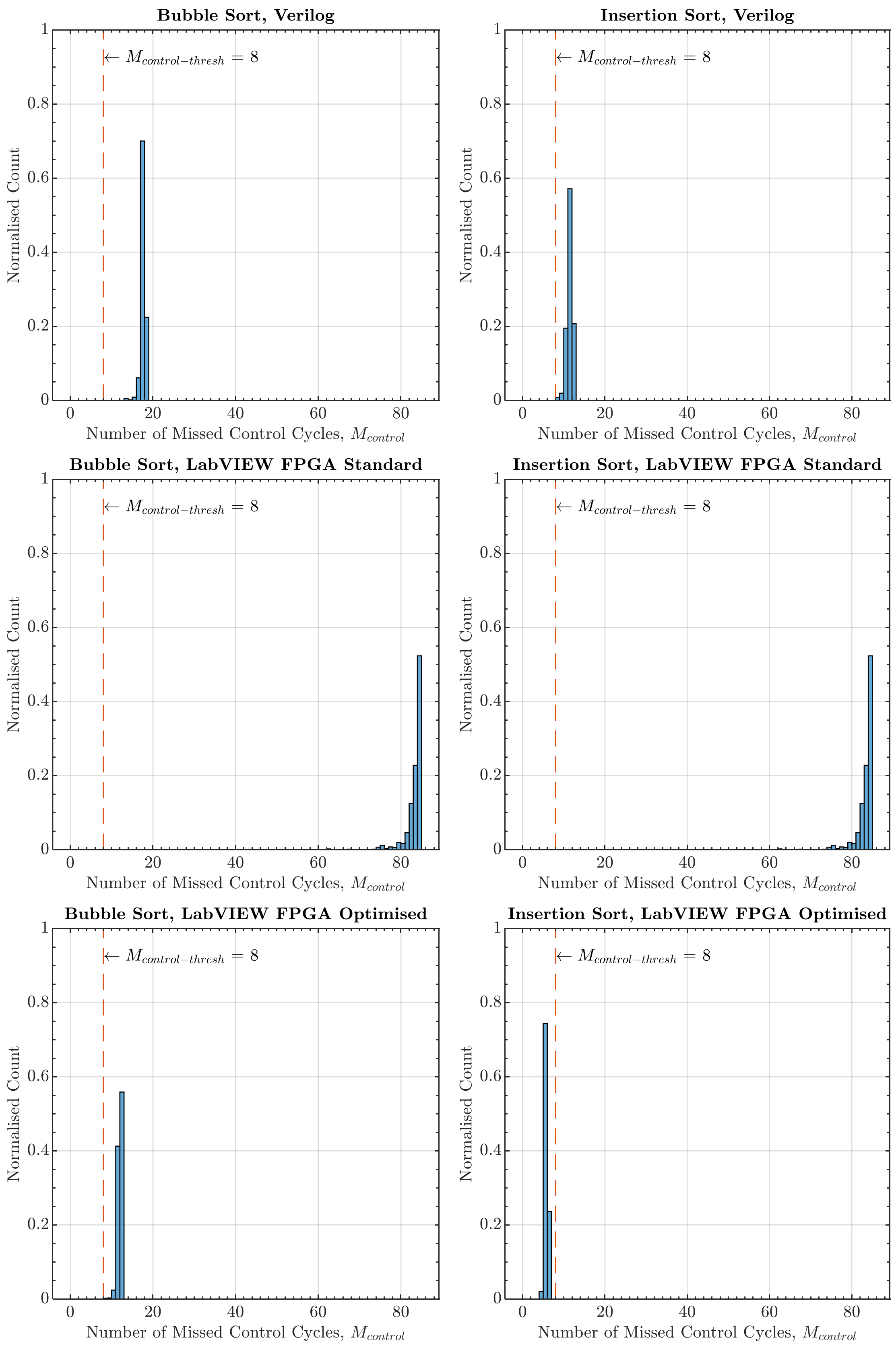


Figure D-14: Sorting algorithm missed control cycle histograms, scenario 2, average tolerance band CBC

Cell Tolerance Band

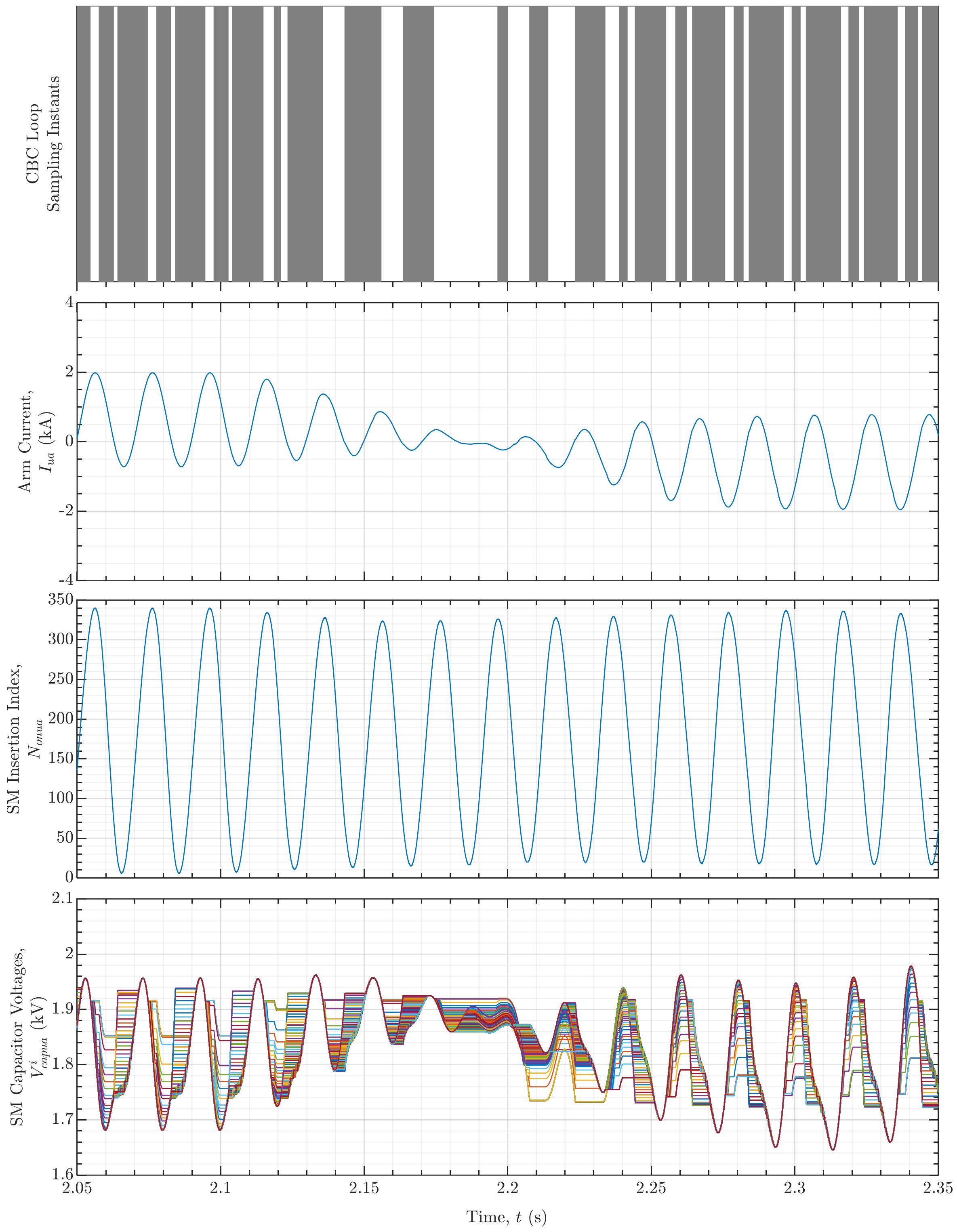


Figure D-15: Upper arm electrical quantities, scenario 2, cell tolerance band CBC

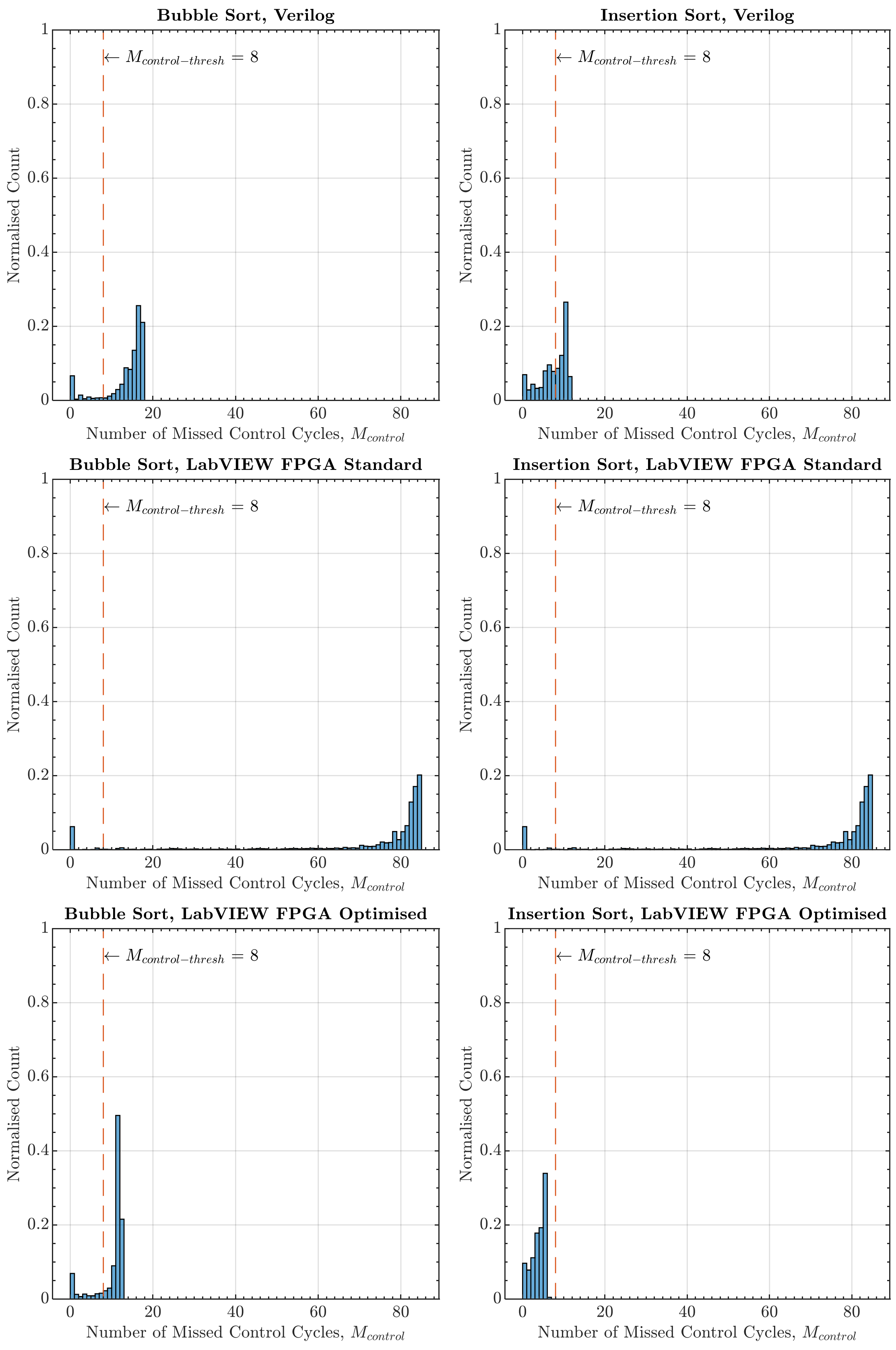


Figure D-16: Sorting algorithm missed control cycle histograms, scenario 2, cell tolerance band CBC

D-3 Scenario 3: AC 3 Phase-ground Fault

Periodic

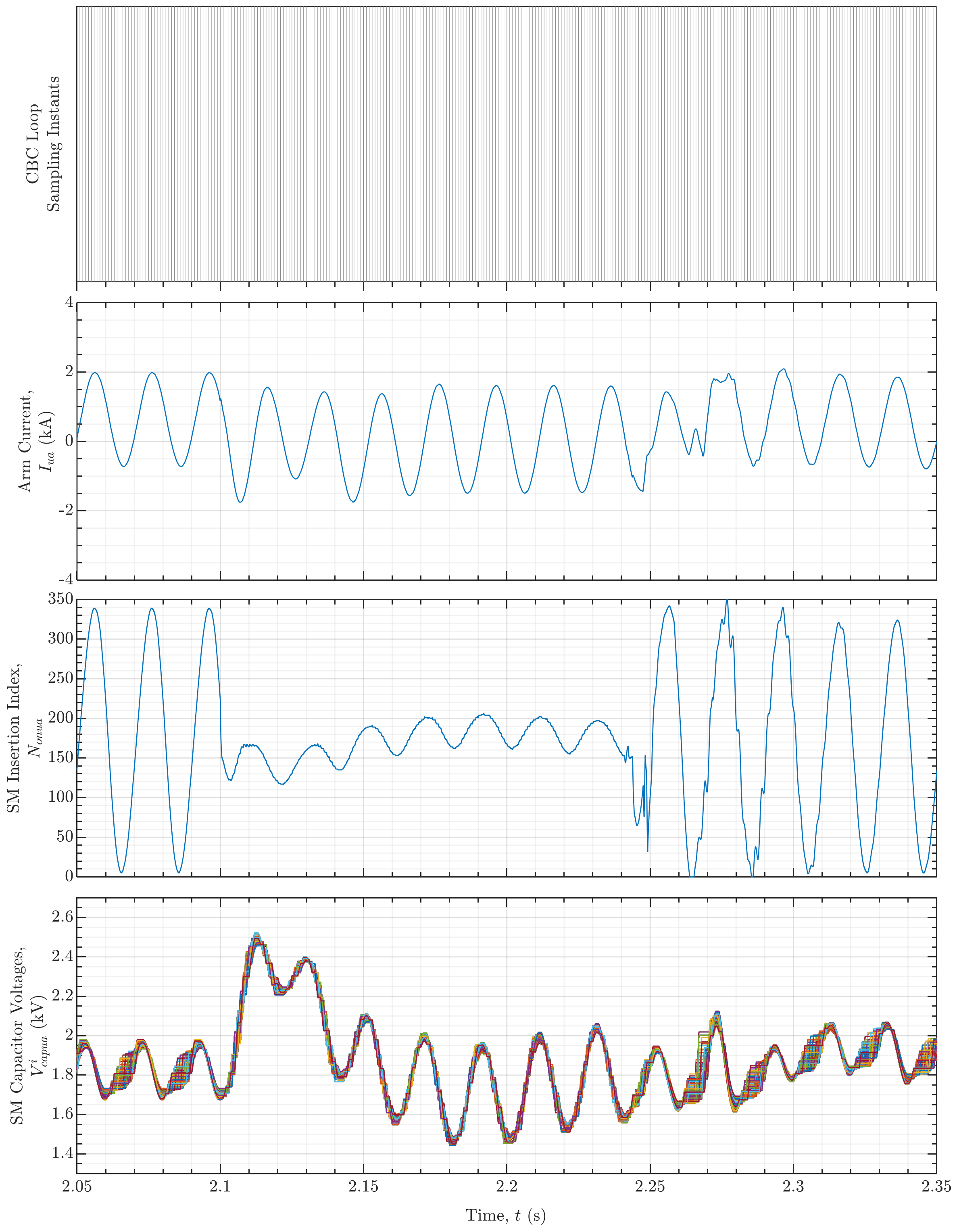


Figure D-17: Upper arm electrical quantities, scenario 3, periodic CBC

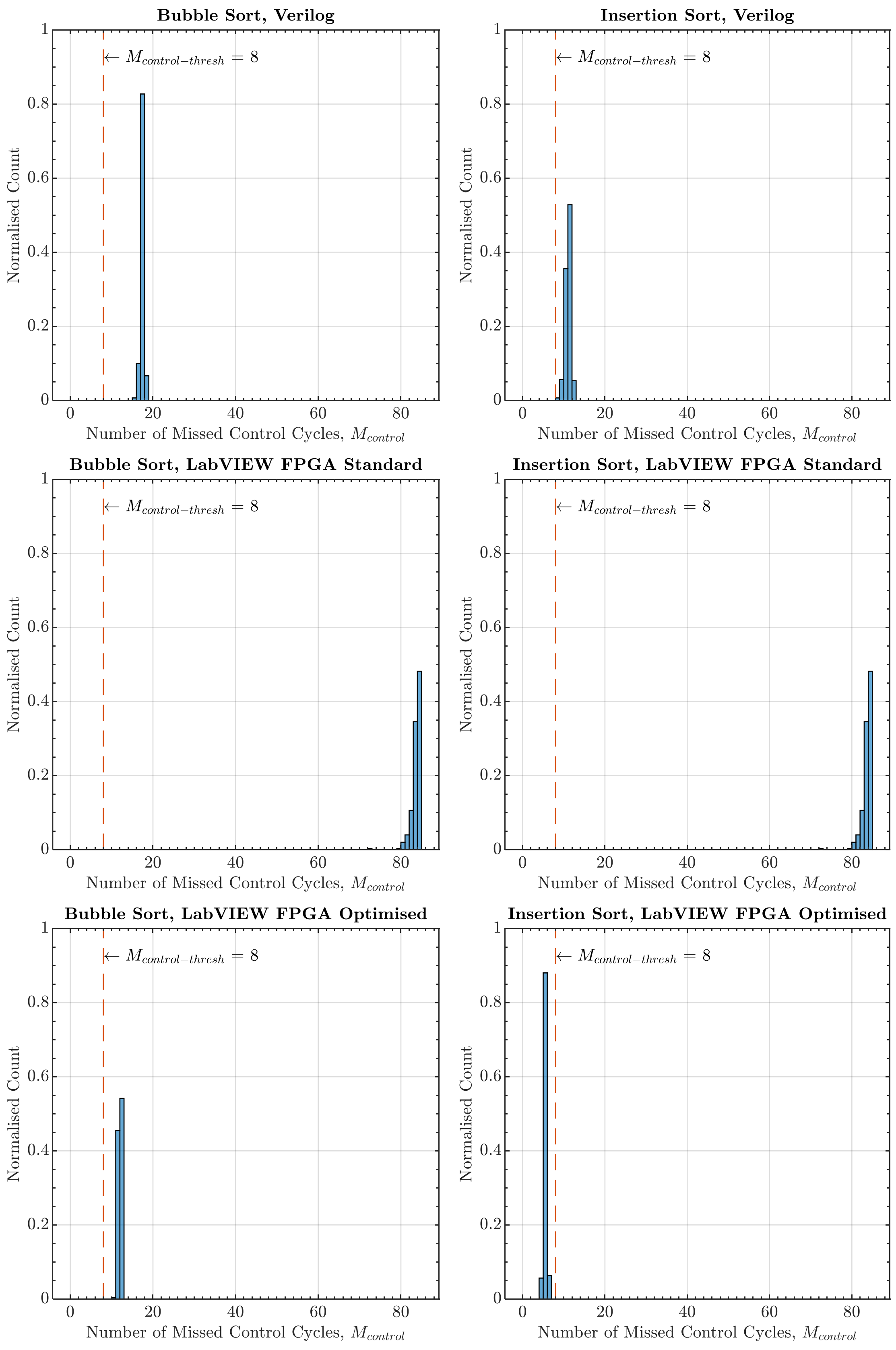


Figure D-18: Sorting algorithm missed control cycle histograms, scenario 3, periodic CBC

Average Tolerance Band

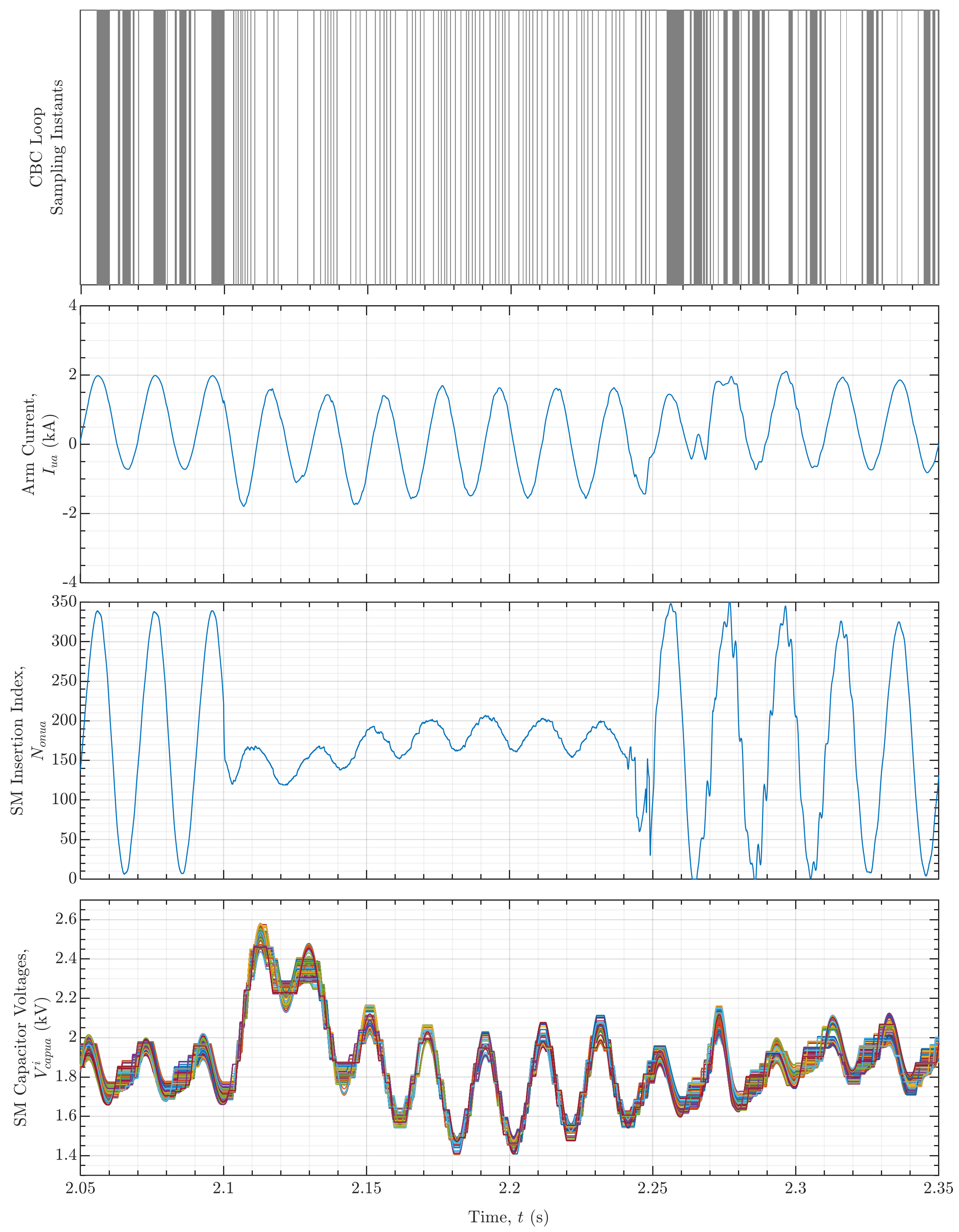


Figure D-19: Upper arm electrical quantities, scenario 3, average tolerance band CBC

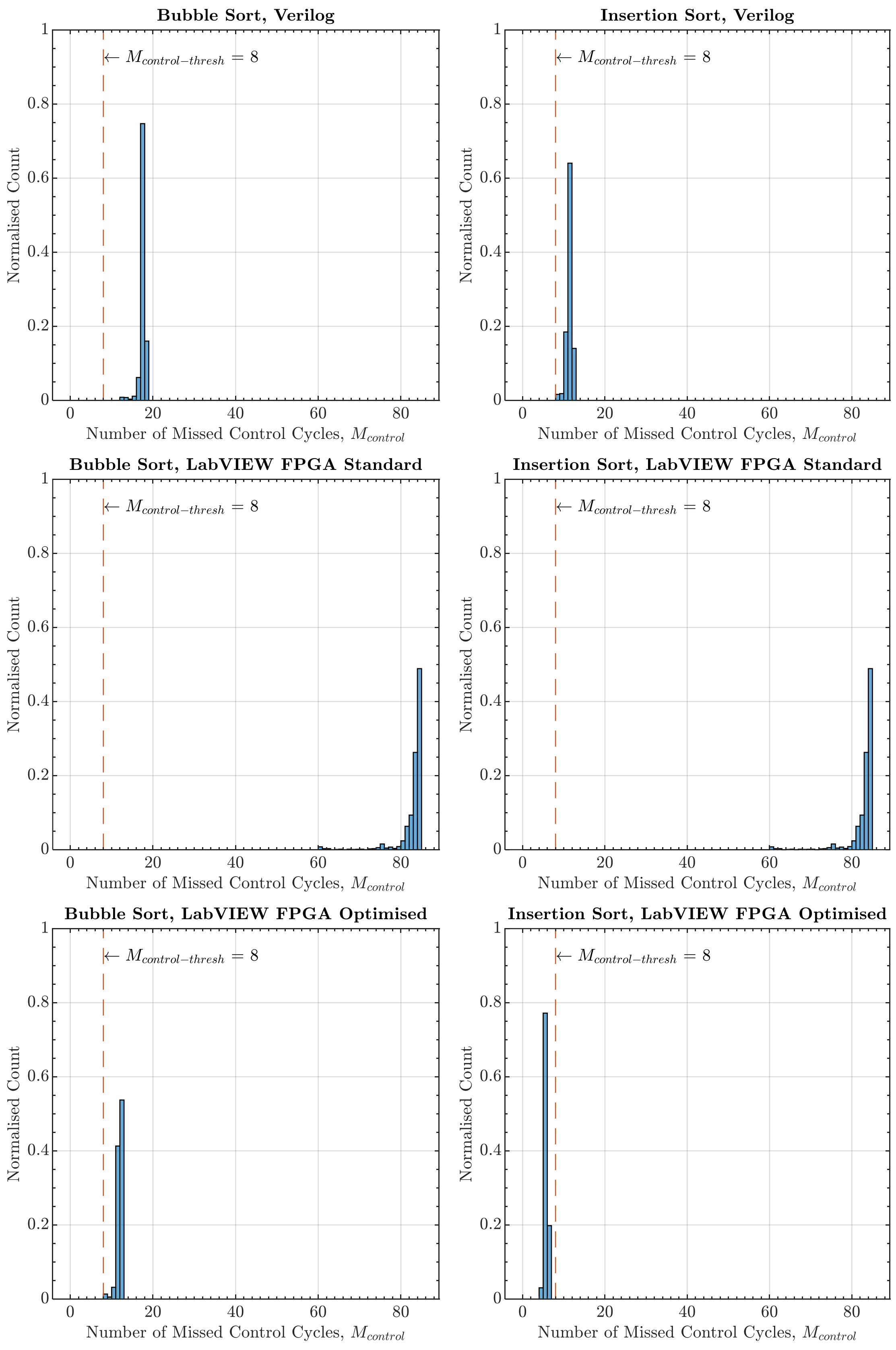


Figure D-20: Sorting algorithm missed control cycle histograms, scenario 3, average tolerance band CBC

Cell Tolerance Band

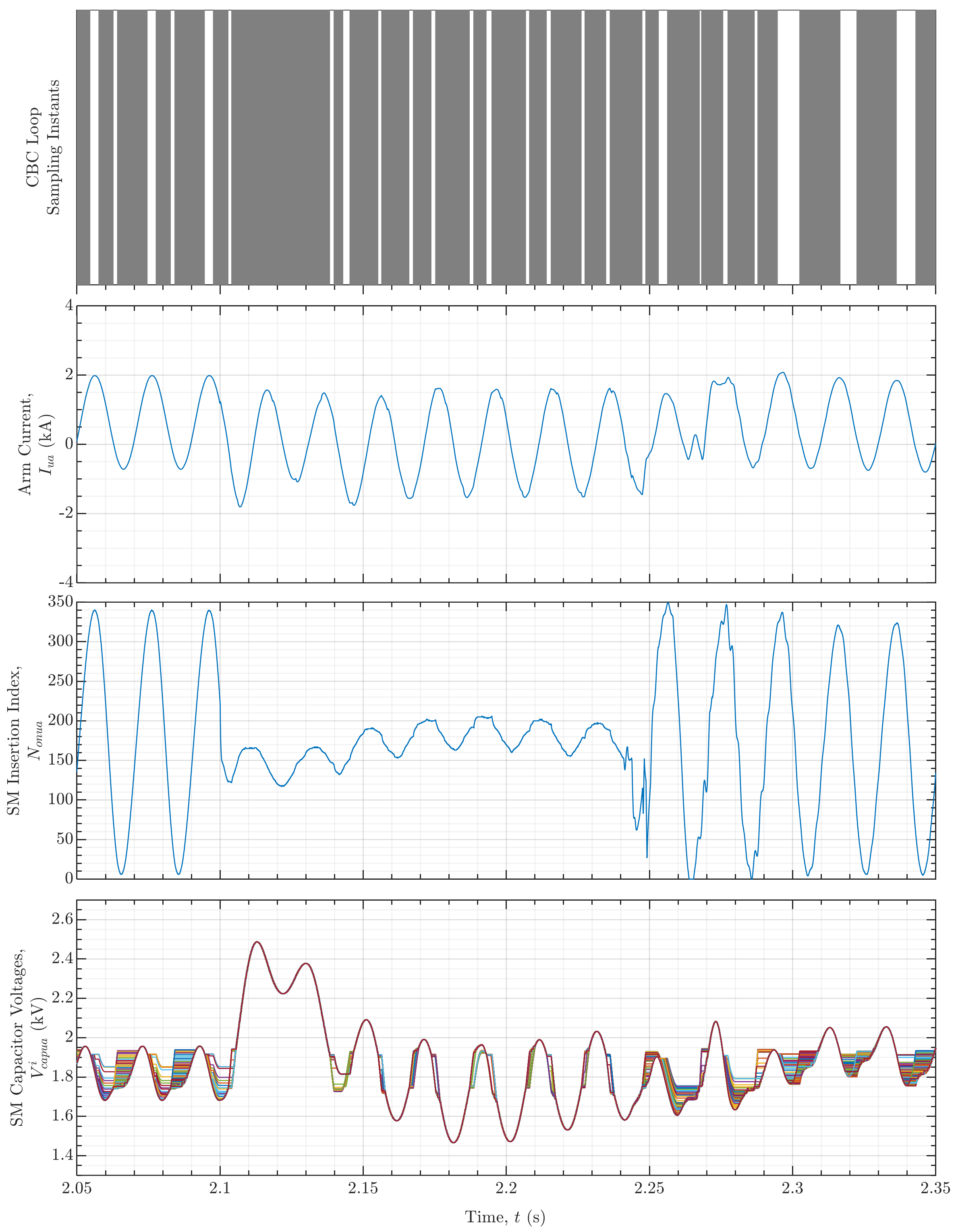


Figure D-21: Upper arm electrical quantities, scenario 3, cell tolerance band CBC

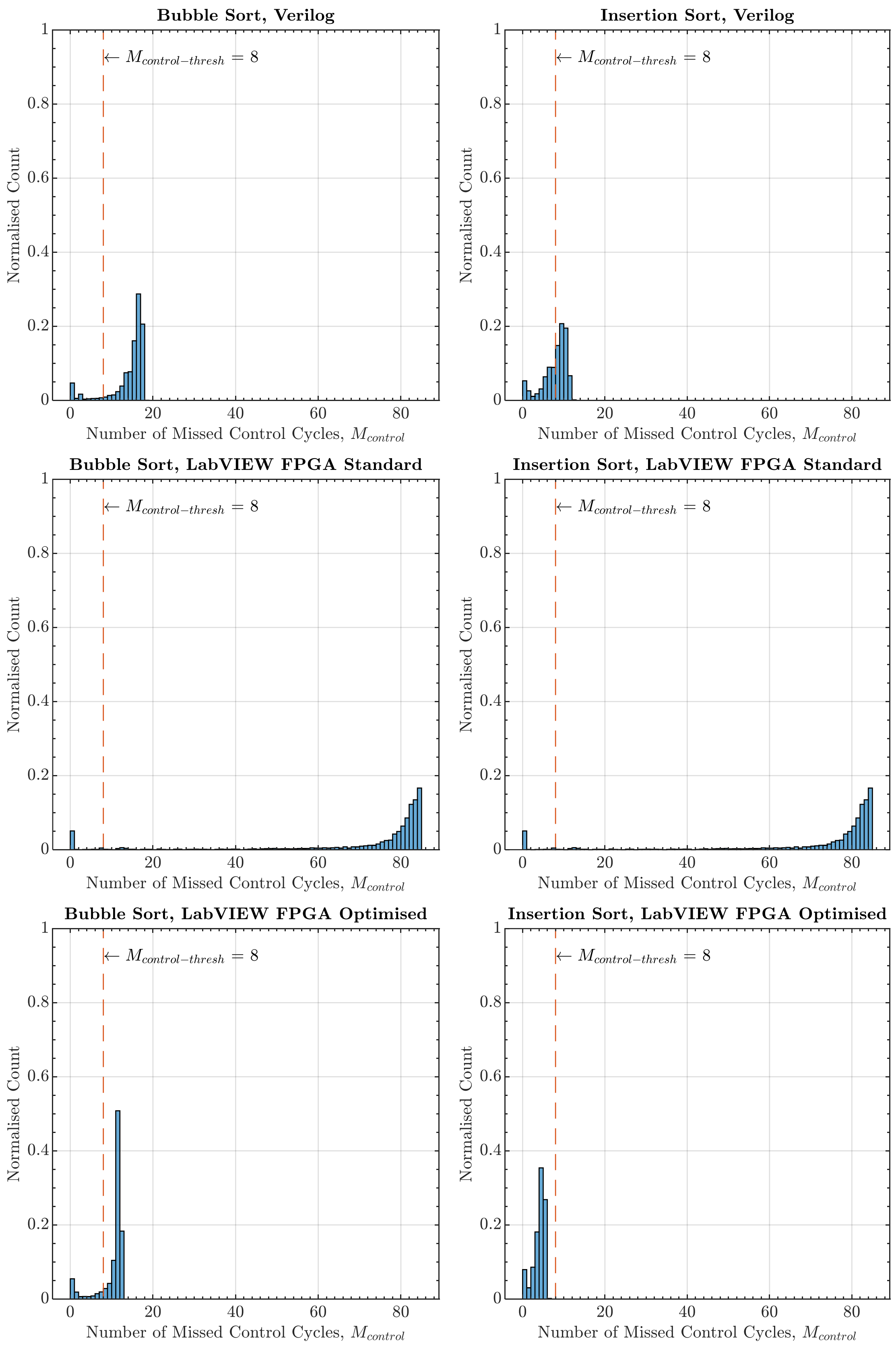


Figure D-22: Sorting algorithm missed control cycle histograms, scenario 3, cell tolerance band CBC

Appendix E - CHP Sensing and Measurement Circuit Modifications

As outlined in Section 7.2.2, some hardware modifications to the converter hardware prototype (CHP) were made during this project to resolve issues first identified in [25]. This appendix provides a more detailed description of the potential sources of measurement noise in the CHP and offers some steps which can be taken to reduce measurement noise when designing future measurement circuits for the CHP. Additional hardware measurements are required and recommended to validate the suggestions included here.

Arm Current Measurement Noise

One of the issues identified was noisy and inaccurate measurements from the Hall-effect current sensors which are used to measure I_{ac}, I_{dc}, and $I_{u,l}$ in the CHP. The arm current measurements are used by the circulating current suppression control (CCSC) and capacitor balancing control (CBC) loops, whilst the AC output current and DC supply current measurements are acquired for logging purposes only. Incorrect arm current measurements will degrade the performance of the CCSC and CBC loops, leading to unrepresentative converter behaviour and possible component damage.

As stated in Section 7.2.2, the voltage output from the Hall-effect sensors on the power interface board (PIB) exhibited a significant noise component. Two possible sources of noise on the Hall-effect sensor outputs were identified in [25] and are explained in more detail below:

- **Common-mode noise on the +5 V power supply bus to the PIB:**
  The PIB shares a +5 V supply bus with the fibre optic breakout board (FOBB) and twelve local controller (LC) field-programmable gate array (FPGA) boards. The circuitry on the FOBB and LC boards is high-speed digital logic operating at clock frequencies typically greater than 1 MHz. Fast switching of digital logic on the FOBB and LC boards produces large dI/dt transients which are coupled onto the local +5 V supply rail on each board. Other circuitry on the LC FPGA boards such as local power supply DC-DC converters will also introduce dI/dt transients. Locally, these transients will cause voltage spikes, which may be coupled onto the main +5 V supply bus due to inadequate local power supply bypassing and long,

high-gauge (therefore high inductance) power supply cables from the +5 V distribution board. The Hall-effect current sensors on the PIB are highly sensitive to noise on the power supply rail. As a result, noise on the +5 V supply bus can be easily coupled into the sensor output if the local power supply rail is not decoupled adequately.

Radiated electromagnetic interference (EMI):
Switching of semiconductor devices in the CHP power electronics (SMs) and ancillary switched-mode power supplies (SMPS) generates EMI. This can be coupled into the Hall-effect sensor measurements in two locations in the measurement circuit:

a. Coupling into the 'primary side' measurand (I_{ac}, I_{dc}, or $I_{u,l}$) due to inductive loops with a large loop area inside the CHP. For example, connections from the six converter arms to the PIB and arm inductors are via long leads (typically > 30 cm), some of which are not tightly paired with their corresponding return current path. This leads to inductive pickup of EMI inside the CHP due to stray inductance caused by a large loop area.
b. Coupling into the 'secondary side' voltage output of the Hall-effect sensors via stray inductance and capacitance in printed circuit board (PCB) traces. Due to the very high (> 10 GΩ [103]) input impedance of the NI PXIe-6363 data acquisition (DAQ) card analogue inputs to which the Hall-effect sensors are connected, the voltage measurement circuit is highly sensitive to noise.

Further troubleshooting during this project also identified a varying DC offset on the Hall-effect sensor outputs which was not caused by a DC current component in the primary side measurand. The DC offset was different for each Hall-effect sensor and varied dependent upon the breaker relay states on the PIB. One potential source for the DC offset was identified as follows:

- **Poor grounding of relay breaker control circuit and Hall-effect sensor circuit:** Several ground references exist in the CHP due to the large number of independent, isolated DC supplies used for the control and communication hardware. For example, the +5 V supply, +24 V supply, and National Instruments (NI) control chassis each have separate, isolated, secondary-side ground references. These ground references are connected together at different (often multiple) points in the

CHP. This can cause ground loop offsets if the grounding scheme is not designed carefully, which manifest themselves as voltage offsets on the output of sensitive measurement circuits, such as the Hall-effect current sensor circuits. The +5 V and NI control chassis grounds are connected together on the PIB and a ground loop between these is one possible cause of the DC offset observed on the current sensor outputs.

The issues with the arm current measurements were eventually resolved by installing six Pico Technology TA189 [97] clamp type current probes in the CHP. The measurements of I_{ac} and I_{dc} are still taken from the PIB Hall-effect sensors, therefore still exhibit the noise and offset issues outlined above.

Measurement Circuit Design Recommendations

As stated previously, the information presented here offers several potential sources of the measurement noise and offset in the Hall-effect sensor outputs in the CHP. Further improvement of the sensing and measurement circuits in the CHP is recommended to resolve the issues identified here. Designing future measurement circuits to ensure that measurements are accurate and noise-free requires and care at all stages of the process. This applies to wire interconnections between circuits, and printed circuit board (PCB) designs for measurement circuits, amongst other components. A wide range of literature is available on design techniques for measurement circuits; however, some general recommendations are listed below:

- Minimise stray inductance in measurement circuits and high-current connections:
  - Use equipment hook-up wire of a suitably high gauge for high-current connections to minimise inductance.
  - Tightly twist current send and return conductors to minimise loop area and therefore stray inductance.
- Shield circuits and conductors which are sensitive to noise.
- Plan and design circuit grounding schemes with care:
  - Identify ground loops and mitigate against these where possible by planning where different ground references will connect.
  - Ensure ground connections have a low stray inductance (short, low-gauge wire or PCB traces).
- Use an independent power supply for sensitive measurement circuits:

    - If this is not possible, ensure that the measurement circuit has adequate local power supply decoupling, such as bypass capacitors of a range of values to target different frequencies.

- Filter DAQ analogue measurements in hardware to attenuate noise outside the expected frequency range of the measurand:
    - This can be done using a simple R-C filter in hardware, or post-acquisition using a filter implemented in software.