Università degli Studi di Padova

Dipartimento di Fisica ed Astronomia "Galileo Galilei"
Dipartimento di Ingegneria dell'Informazione
Corso di Laurea in Fisica

Generazione di numeri casuali basata sulla rivelazione di singoli fotoni

Relatore:
Giuseppe Vallone

Laureando:
Leonardo Canevarolo

Correlatore:
Davide G. Marangon

Anno Accademico 2015/2016

Indice

Elenco delle figure

Elenco delle tabelle

Abstract

La tesi verterà sulla generazione di numeri casuali, tecnica che trova utilizzo in molteplici settori del sapere, dalla crittografia, alle simulazioni numeriche, in tutti i contesti di ricerca industriale e scientifica, e in tutte quelle situazioni in cui risulta necessario avere a disposizione un set di dati che varino in maniera imprevedibile a priori. Si provvederà perciò a presentare lo stato dell'arte della generazione di numeri casuali, distinguendo in particolar modo tra Random Number Generator "veri" (TRNG) e pseudo tali, mostrandone i punti di forza e i punti di deboli di ciascuno di essi e focalizzandosi infine sui primi. Dopo aver introdotto alcuni concetti teorici, si presenterà il Quantum Random Number Generator utilizzato durante l'attività di laboratorio e i metodi implementati per estrarre bit casuali da esso, determinando quale tra essi risulti essere il più efficiente e in grado di produrre output con proprietà statistiche migliori.

Capitolo 1

Introduzione

Negli ultimi anni uno dei campi che ha destato maggiore interesse, indirizzando la ricerca a livello mondiale, è stato quello dei generatori di numeri casuali. Il motivo di tale esplosione può essere desunto dalla richiesta sempre crescente di dati totalmente imprevedibili a priori, per via del loro utilizzo in numerosi ambiti come quello della ricerca scientifica, della simulazione numerica o del gioco d'azzardo. Tuttavia il vero motore di tale fervore scientifico può essere ricondotto alla crittografia e alle numerose sue applicazioni che utilizziamo ogni giorno e che vanno dai protocolli per la comunicazione sicura sia su Internet che su rete mobile, fino all'accesso alle reti wireless e a molti applicativi di encrypting. Poiché molti di questi si basano sul principio di Kerckhoff, il quale asserisce che un sistema crittografico dev'essere sicuro anche quando il protocollo di comunicazione viene reso pubblico, la loro sicurezza deriva dalla segretezza della chiave creata per la comunicazione dai due interlocutori, che deve pertanto essere totalmente imprevedibile dall'esterno. La necessità di tali dati richiede che la loro casualitá sia certificabile. Per adempiere a tale richiesta sono state sviluppate numerose tecniche che si sono rivelate avere ciascuna i propri vantaggi e difetti. Soltanto negli ultimi anni tuttavia si è iniziato a produrre generatori la cui randomicità fosse provabile. Si presenterà pertanto lo stato attuale dell'arte nella generazione dei numeri casuali e il problema cui si è cercato di rispondere con la presente tesi.

1.1 Generatori di numeri pseudo-casuali

Storicamente per ovviare al problema della generazione di numeri casuali si sono perseguiti due approcci differenti in maniera sostanziale: l'approccio computazionale e quello fisico; il primo ha portato allo sviluppo degli Pseudo-Random Number Generator, i generatori di numeri casuali basati sull'implementazione informatica di processi complessi tali da costruire sequenze di cifre lunghe a piacere, in una maniera deterministica completamente individuata dal valore iniziale a loro attribuito. A causa di tale intrinseca prevedibilità, i PNRG sono detti randomici soltanto in apparenza e permettono di ottenere una soluzione esclusivamente in contesti che non richiedano una vera randomicità. Essi producono output con buone proprietà statistiche, sono rapidi e facilmente implementabili (richiedono soltanto un computer per essere utilizzati). Per contro, presentano una forte correlazione a distanza che si ripercuote sulla sicurezza delle chiavi crittografiche generate con essi (si veda l'esempio della rottura della sicurezza del Netscape Browser nel 1996) e in comportamenti inaspettati dei metodi di Monte Carlo che li utilizzano [1].

1.2 "Veri" generatori di numeri casuali

L'approccio alternativo ai PRNG sfrutta le misurazioni di fenomeni aleatori (o considerati tali per la loro elevata complessità) per produrre dati randomici. Tali RNG sono detti True Random Number Generators e ciascuno di essi contiene due blocchi separati: una sorgente di entropia ed un blocco di post-processing; la prima è un sistema fisico a cui può essere associata una variabile aleatoria da cui poter estrarre dei dati casuali e il secondo

un algoritmo in grado di distillare la loro randomicità e di ovviare ad eventuali problemi sperimentali che andrebbero ad inficiare la qualità dei risultati.
Naturalmente tale definizione comporta che ogni fenomeno difficilmente prevedibile possa diventare un TRNG.
Ne è un esempio il RNG basato sulle misurazioni in microsecondi degli intervalli di tempo che intercorrono tra due utilizzi consecutivi della barra spaziatrice da parte di un utente in una sessione al computer, che produce uno zero o uno a seconda se il numero associato alla misura è pari o dispari, ottenendo in tal modo dei bit casuali. Ciò che si dimostra sperimentalmente è che molti utenti finiscono il 70% delle volte con il far trascorrere un numero di microsecondi pari tra due colpi di barra spaziatrice, il che si traduce, per definizione del metodo, in un numero maggiore di zeri presenti nella sequenza di output rispetto al numero degli uno. Questa, che sembrerebbe una peculiarità innocua, è in realtà uno dei principali svantaggi dei TRNG, in quanto determina un discostamento delle stringhe prodotte dalle proprietà necessarie per la randomicità, richiedendo così inevitabilmente un cosiddetto trattamento di debiasing in fase di post-processing (l'eliminazione a monte di tale problema necessiterebbe di una calibrazione accurata, un'operazione difficile da fare per via dell'enorme mole di tempo richiesto).
Quello sovraesposto è un esempio di RNG non-deterministico non-fisico, ma la famiglia di TRNG è ben più vasta ed è stata schematizzata nel diagramma riportato di seguito:

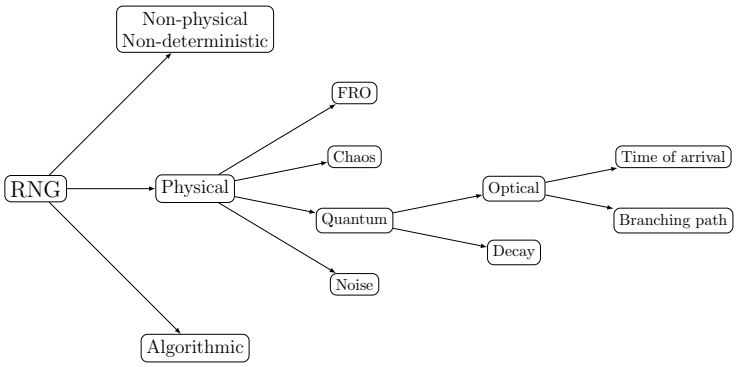


Figura 1.1: Diagramma di tutte le tipologie di RNG già esistenti

La classe di TRNG maggiormente importante è quella degli Hardware RNG, i generatori di numeri casuali fisici che sfruttano come sorgente di entropia processi non-deterministici fisici. Molto spesso manca uno studio rigoroso dell'aleatorietà degli HWRNG implementati nel corso degli anni perché si è assunto che la complessità dei fenomeni soggiacenti li rendesse automaticamente degli RNG [2]. Tuttavia, poiché qualsiasi sistema basato sulla meccanica classica evolve, in linea di principio, in maniera prevedibile e deterministica, tale conclusione è risultata spesso quantomeno discutibile. Per ovviare a tale problema, a partire dagli anni '70 si è iniziato a sfruttare la natura probabilistica della meccanica quantistica, producendo i primi Quantum Random Number Generators.

1.3 Quantum Random Number Generators

Data l'ampia gamma di fenomeni quantistici intrinsecamente probabilistici, la classe dei QRNG è molto vasta e si suddivide principalmente in due categorie: gli RNG basati sui decadimenti radioattivi (i primi ad essere studiati, ma rapidamente abbandonati per via della loro scarsa maneggevolezza e potenziale pericolosità) e gli Optical QRNG, che rapidamente hanno sostituito i primi per via della maggior praticità della produzione di fotoni tramite laser e LED, oltre che per il loro rate di produzione di eventi generalmente più alto. Gli OQRNG a loro volta si suddividono in numerose sottoclassi a seconda dell'osservabile fondamentale usata nelle diverse implementazioni che può andare dalla polarizzazione dei fotoni, al loro tempo di arrivo, alle loro fasi. Le due principali tipologie di OQRNG sono i generatori spaziali e i generatori temporali, che verranno ora brevemente presentati.

1.3.1 Branching path RNG

I branching path RNG generano bit casuali andando a rilevare come vengano proiettati in seguito a delle misurazioni dei qubit, la più piccola quantità di informazione esistente che può essere descritta come la combinazione lineare dei due bit '0' e '1' e che può essere trasportata da fotoni singoli. I generatori di questo tipo consistono infatti di una sorgente di luce polarizzata circolarmente, un polarization beam splitter e due rivelatori posti come in figura (1.2), che determinano il percorso fatto dai fotoni dopo aver cambiato forzatamente polarizzazione in maniera casuale tramite il beam splitter.

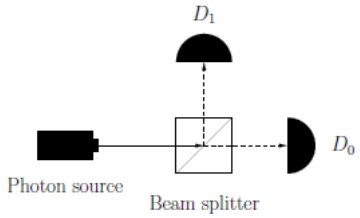


Figura 1.2: Schema di un tipico Branching Path RNG. D_1 e D_0 rappresentano i due detector. Il primo produce un '1' ogniqualvolta il fotone uscente dal beam splitter ha polarizzazione orizzontale, il secondo genera uno '0' nel caso complementare.

Poiché la polarizzazione iniziale dei fotoni contiene egualmente la polarizzazione lineare verticale e quella orizzontale, e poiché i fotoni non possono scindersi a metà, essi hanno sempre eguale probabilità di percorrere i due percorsi fino ai due rivelatori; se si associa ad una rivelazione del primo il bit '0' e il bit '1' alla rivelazione dell'altro, si ottiene un perfetto RNG (a meno di limitazioni sperimentali che ne peggiorano l'output) la cui randomicità diventa provabile perchè insita nelle leggi della meccanica quantistica.

1.3.2 Time of arrival RNG

I time of arrival RNG solitamente sono composti da una sorgente debole di fotoni, un rilevatore e un apparato elettronico in grado di registrare, a seconda del modello usato, gli istanti di rilevazione o il numero di eventi rilevati in un fissato periodo di tempo. Indicato con λ il numero medio di fotoni emessi dalla rispettiva sorgente per unità di tempo (almeno se λ risulta essere sufficientemente basso), nel primo caso essi producono bit casuali dalla distribuzione esponenziale dei tempi di rilevazione, data da $\lambda e^{-\lambda t}$; nel secondo caso invece dalla probabilità di registrare n eventi in un periodo di tempo T, descritta dalla distribuzione di Poisson

$$P(n) = \frac{(\lambda T)^n}{n!} e^{-\lambda T}$$

Per farlo essi chiaramente necessitano della misurazione dei tempi. Il modo più utilizzato per compiere questa operazione in maniera digitalizzata è quella di discretizzare gli intervalli temporali frapposti tra due eventi consecutivi, misurando il numero di edge (il numero di volte in cui un clock di riferimento passa dallo stato logico 'Low' allo stato logico 'High') che intercorrono tra le due rivelazioni. Un esempio di applicazione del metodo è rappresentato in figura (1.3).

1.3.3 Importanza del confronto

Per fare maggiore chiarezza nella pletora di realizzazioni di OQRNG già esistenti risulta necessario una loro classificazione in termini di efficienza e bontà statistica dei loro output per determinare quale blocco di post-processing convenga in generale utilizzare. A tale scopo si utilizzerà un QRNG tale da poter applicare tre metodi di post-processing diversi e se ne confronteranno i risultati.

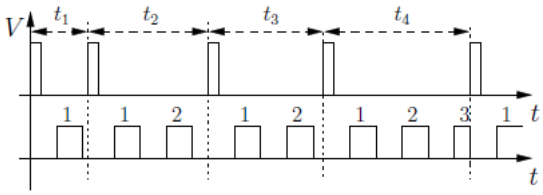


Figura 1.3: Esempio della raccolta dei tempi di arrivo di cinque eventi. Le tensioni lungo l'asse delle ordinate rappresentano nel primo sistema gli impulsi generati dalle rivelazioni e nel secondo le pulsazioni del clock di riferimento. Si noti che i segnali che si trovano nello stato 'High' in corrispondenza di una nuova rivelazione vengono "abbassati" forzatamente allo stato più basso. Tale metodo viene detto con resettable clock ed è stato proposto per evitare difetti statistici nelle stringhe di bit prodotte.

Capitolo 2

L'apparato sperimentale

Il time of arrival QRNG utilizzato per l'attività di laboratorio è il LinoSPAD del TU Delft, un dispositivo elettronico costituito da 256 CMOS-SPAD (Single Photon-Avalanche Diode) connessi allo Xilinx Spartan 6 FPGA.
Lo schema dell'apparato è il seguente:

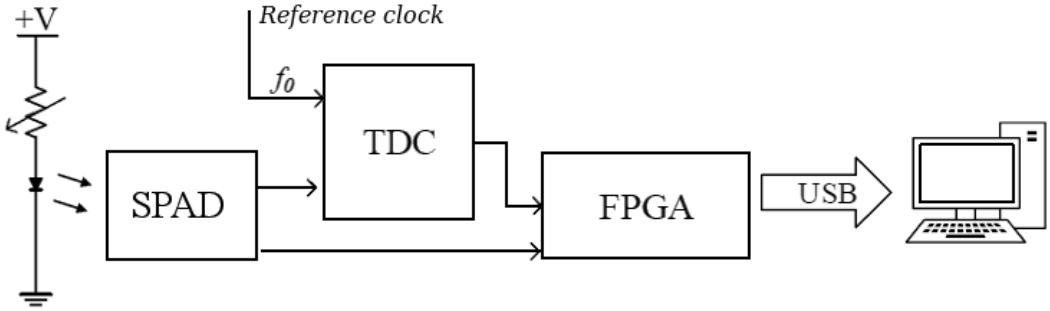


Figura 2.1: Parametri utilizzati: $f_0 = 400\,\text{MHz}$, $V = 1.5\,\text{V}$

L'FPGA è un circuito integrato che controlla le funzionalità degli SPAD (detti pixel), si interfaccia con il computer tramite un cavo USB e contiene 64 Time-to-Digital Converter, che permettono di registrare i tempi di arrivo (detti tag) dei fotoni sul dispositivo. Il funzionamento dei TDC è simile a quello riportato in generale per i time of arrival QRNG, però si differenzia da essi perché essi non necessitano di due rilevazioni per ottenere un singolo tag temporale. Ciascun TDC infatti è collegato ad un reference clock da 400 MHz e campiona una delay line da 140 bit ogni 2.5 ns, associando un numero che varia da 0 a 139 per indicare il bin temporale in cui riceve l'impulso elettrico prodotto da uno SPAD dopo una rivelazione. L'informazione viene poi passata all'FPGA che determina in maniera cumulativa l'istante di arrivo del fotone. La risoluzione sperimentale è pertanto stimabile nella lunghezza temporale associata ad un bin della delay line, ovvero a $\frac{2.5ns}{140} \approx 17.86\,\text{ps}$.
Ogni TDC può registrare al più il tempo di arrivo di 512 eventi. Poiché un superamento di tale soglia comporta una sovrascrittura dei dati iniziali, viene data la possibilità di modificare la finestra temporale di integrazione in maniera tale da non porsi in tale condizione sperimentale e di ovviare alla scarsità di dati ripetendo più volte l'acquisizione, organizzando i risultati in 'frame' da 512 elementi. Ogni presa dati consta pertanto di un vettore da $512 \cdot n_{frames}$ elementi contenente tutti i tag temporali degli eventi rilevati.
Il LinoSPAD viene dato in dotazione assieme ad un software che permette di gestire la raccolta dati.

2.1 Limitazioni sperimentali degli SPAD

Come ogni dispositivo non-ideale, il LinoSPAD risente di alcune problematiche sperimentali. Le principali sono le seguenti:

2.1.1 Deadtime

Il deadtime rappresenta il tempo necessario al dispositivo a ripristinare le sue condizioni interne iniziali dopo una rivelazione. Esso viene stimato in 40 ns e comporta che nessun evento in tale intervallo temporale possa essere registrato.

2.1.2 Afterpulsing

Un afterpulse è un impulso che si genera nel dispositivo dopo una rivelazione e che determina la registrazione di un evento artefatto in aggiunta all'evento associato alla rivelazione reale. Dal punto di vista fisico esso viene causato da dei portatori di carica intrappolati all'interno del diodo in uno stato metastabile. Ciò permette di definire un tempo di vita medio massimo τ_{ap} passato il quale la probabilità di rilevare un afterpulse è più piccola di una fissata soglia. L'effetto netto della presenza degli afterpulse è l'aumento della probabilità di rilevare due eventi separati per un tempo $t < \tau_{ap}$, che si traduce in una deformazione (mostrata in figura (2.2)) della distribuzione di probabilità esponenziale dei possibili valori di t. In generale non è possibile definire la forma analitica della densità discreta di probabilità degli afterpulse, tuttavia tutti i modelli fatti [3] assumono che essa sia una funzione che decresce velocemente (esponenzialmente o come una somma di esponenziali) all'aumentare del tempo. Ciononostante risulta sempre possibile stimare un limite superiore della probabilità di afterpulsing e il tempo di vita medio massimo τ_{ap}. Nel caso del LinoSPAD si trova che $\tau_{ap} = 400\,\text{ns}$ e la probabilità di afterpulsing in tale finestra è pari a 0.12.
Tale fenomeno inficia l'indipendenza degli eventi registrati dal dispositivo e deve pertanto essere trattato appositamente nella fase di post-processing.

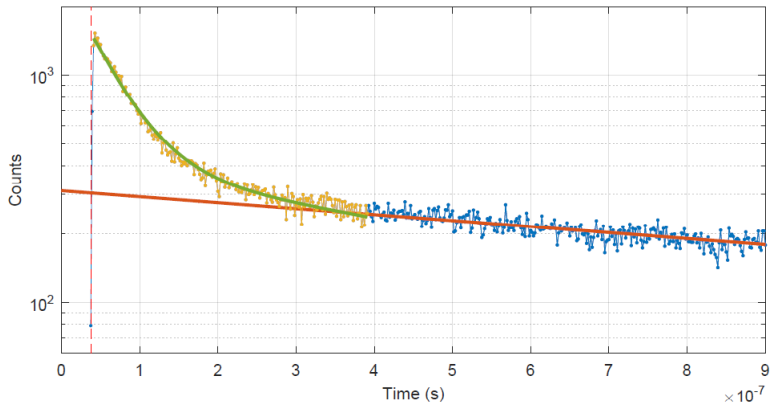


Figura 2.2: Istogramma in scala logaritmica che rappresenta un tipico andamento delle differenze temporali degli eventi registrati dal LinoSPAD.
Si noti come il deadtime determini l'azzeramento dell'istogramma fino a circa 40 ns. Il discostamento dall'andamento esponenziale fino a 400 ns è un'evidenza dell'effetto di afterpulsing.

2.1.3 Dark count

I 256 pixel del LinoSPAD non sono perfettamente equivalenti dal punto di vista del funzionamento interno. Oscurando il dispositivo, è possibile infatti vedere come in condizioni di luminosità molto bassa essi registrino un numero di eventi che varia molto e che può essere anche molto elevato. Determinando i picchi dei conteggi ottenuti in tali condizioni (detti dark count), è possibile individuare tre categorie di pixel, indicate per comodità come LLP (Least Luminous Pixels), NLP (Normal Luminous Pixels), MLP (Most Luminous Pixels). Tale classificazione viene considerata una caratterizzazione del dispositivo in quanto non

varia a seconda della presa dati. Si riporta pertanto la "mappa" dei pixel, colorata in modo da visualizzare con un colore più caldo gli SPAD maggiormente luminosi.

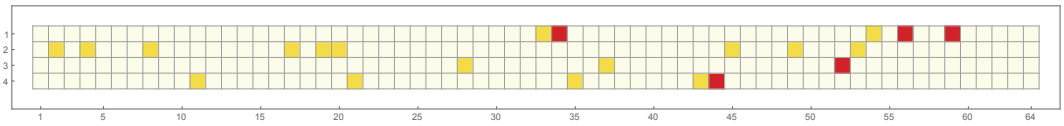


Figura 2.3: Mappa delle intensità dei pixel del dispositivo.
Si noti come vi siano solo 5 MLP (in colore rosso, in corrispondenza dei pixel 34,56,59,180 e 236) e 17 NLP. Data la loro peculiarità, tutti questi pixel sono stati scelti per l'analisi dati.

Capitolo 3

Prerequisiti teorici

Per poter analizzare i risultati dell'analisi dati in maniera adeguata, verranno ora esposte le definizioni e i significati fisico-matematici di tutte le nozioni utilizzate.

3.1 L'entropia di Shannon

In teoria dell'informazione, per poter parlare rigorosamente della trasmissione e della memorizzazione di dati, è di fondamentale importanza la definizione di informazione per rendere la materia quantificabile. Per introdurre tale concetto, si consideri una sorgente discreta e stazionaria tale da emettere una sequenza X composta di simboli x_i appartenenti ad un alfabeto di cardinalità L, ognuno contraddistinto da una data probabilità di emissione p_i. Si definisce come misura di informazione la quantità

$$I(x_i) = \log_b(\frac{1}{p_i})$$

che si misura in *bit* e risulta essere legata all'incertezza relativa al simbolo x_i nella sequenza X: maggiore è la probabilità che esso vi compaia, minore è l'informazione che tale simbolo contiene.
Claude Shannon, nel 1948, definì l'entropia di una sequenza X come la media pesata delle informazioni dei diversi simboli contenuti in essa, ovvero come:

$$H(X) = -\sum_{i=0}^{n} p_i \log_b(p_i)$$

Essa si misura in *bit/simbolo* e rappresenta il tasso medio di informazione per simbolo contenuto nella sequenza.
L'importanza di tale grandezza tuttavia deriva dal fatto che essa quantifichi il livello di casualità di X. Per ottenere una spiegazione intuitiva di tale proprietà, si consideri un alfabeto composto dai soli simboli $\{0, 1\}$. In tal caso l'entropia appartiene sempre all'intervallo $[0; 1]$ e raggiunge il valore massimo in corrispondenza dell'equiprobabilità dei bit, mentre si azzera quando la sequenza non contiene alcuna informazione, ovvero nel caso in cui si abbia $p_i = 1$ per uno dei due indici.
Operativamente, tramite il primo teorema di Shannon sulla codifica di sorgente, è possibile definire l'entropia come il massimo rate di compressione di una sequenza infinite di variabili aleatori identicamente e indipendentemente distribuite che non causi una perdita di informazione.
Nelle applicazioni infine è spesso utile il concetto di entropia minima, definita come

$$H_\infty(X) = -\max_i \log_b(p_i)$$

in quanto consente di determinare la minima quantità di informazione disponibile nel peggiore dei casi.

3.2 Le procedure estrattive

Sia $\Omega = \bigcup_{n=0}^{\infty} \{0,1\}^n$ l'insieme di tutte le stringhe di lunghezza arbitraria composte da bit. Siano $u, v \in \Omega$ due stringhe. Diremo che u è un prefisso di v se $u \leq v$, cioè se v contiene u. Diremo che una mappa ψ: $\Omega \to \Omega$ è una procedura estrattiva se

- $\forall u, v \in \Omega$ tali che $u \leq v$ si ha $\psi(u) \leq \psi(v)$
- se $\forall x \in \Omega$ tale che i suoi bit casuali siano distribuiti indipendentemente secondo (p, q), allora $\psi(x)$ è distribuita uniformemente in $\{0,1\}^k$ ogniqualvolta $l(\psi(x)) = k$

Tale definizione classifica le procedure estrattive come tutte le funzioni in grado di distillare una stringa con un eguale numero di uno e di zeri a partire da una stringa affetta da bias. Una delle loro principali caratteristiche è il rate, che si definisce come la funzione

$$r(p) = \limsup_{n \to \infty} \frac{E\left[l(\psi(x))\right]}{n}$$

dove $x \in \Omega$ è una stringa di bit indipendenti distribuita secondo (p,q), $E[\cdot]$ denota il valore di aspettazione e l è la funzione che restituisce la lunghezza della stringa $\psi(x)$.
Esso quantifica il numero di bit estratti in media dalla procedura, dando un'idea della sua efficienza.

3.3 Proprietà statistiche fondamentali per l'analisi delle stringhe

3.3.1 Bias

Definiamo come bias di una sequenza X la grandezza

$$b(X) = 1/2 - p(1)$$

dove $p(1)$ è la probabilità associata al bit '1'. Tale grandezza indica quanto gli elementi della stringa X si discostano dalla condizione di equiprobabilità.

3.3.2 Autocorrelazione

Una grandezza molto importante per l'analisi delle stringhe prodotte è l'autocorrelazione, che permette di stabilire se vi sono delle ripetizioni interne ad esse che potrebbero aumentarne la prevedibilità e la non-randomicità. Essa è descritta da una serie di coefficienti a_k, che specificano per un dato lag k il grado di correlazione della stringa con la stringa stessa traslata indietro di k posizioni. Data una sequenza di bit $Y_1 \dots Y_N$, il valore dei coefficienti a_k viene posto pari a

$$a_k = \frac{\sum_{i=1}^{N-k}(Y_i - \bar{Y})(Y_{i+k} - \bar{Y})}{\sum_{i=1}^{N}(Y_i - \bar{Y})^2}$$

Per definizione vale $a_k \in [-1; 1]\, \forall k$. Quando $a_k = 1$ si dice che la stringa è perfettamente correlata con se stessa traslata di k, mentre quando $a_k = -1$ le due stringhe si dicono anti-correlate. Come nella correlazione di Pearson, il segno dei coefficienti è positivo se i valori assunti dalla stringa traslata sono più alti della media quando anche i valori assunti della stringa originale lo sono (e negativo nel caso contrario). Si vede subito che ciò significa che un coefficiente è positivo se per un dato k la stringa traslata presenta frequentemente bit dello stesso tipo della stringa originale.

3.4 Bit casuali genuini

In generale, diremo che una sequenza di variabili aleatorie $X_1, \dots, X_n$ è distribuita identicamente e indipendentemente se tutte le X_i sono statisticamente indipendenti tra di loro

e seguono la stessa distribuzione di probabilità. In particolare, ciò consente di definire il concetto di sequenza di bit genuini come una stringa in cui tutti i bit sono identicamente e indipendentemente distribuiti. Si noti come tale richiesta sia più forte della sola uniformità in quanto stringhe senza bias possono essere fortemente correlate.
Una generalizzazione pratica della precedente definizione è fornita dal concetto di variabili aleatorie interscambiabili, secondo la quale, data una sequenza di variabili aleatorie $X_1, \ldots, X_n$, esse sono interscambiabili se vale la proprietà

$$P(X_1 X_2 \ldots X_n) = P(X_{\sigma(1)} X_{\sigma(2)} \ldots X_{\sigma(n)}) \quad \forall \sigma \in \Sigma_n$$

3.5 Test d'ipotesi per gli RNG

Nel seguito verranno utilizzati spesso i test di ipotesi per controllare la validità dei valori ottenuti dall'analisi. Essi si basano sull'ipotesi nulla H_0 che i dati a cui vengono applicati provengano da un RNG ideale e determinano la probabilità che un RNG ideale dia un risultato "peggiore" . Poiché le grandezze testate sono (o possono essere messe in corrispondenza [7]) con variabili gaussiane, ciò significa che i test integrano le code della gaussiana a partire dal valore testato e valutano se il risultato di tale operazione resta sopra una prefissata soglia di accettabilità, misurando così quanto il valore iniziale fosse lontano dal centroide della gaussiana. Siccome i test di ipotesi non possono avvalorare l'ipotesi nulla, essi possono essere utilizzati soltanto per assicurarsi che le diverse grandezze testate non si discostino eccessivamente da quelli che un RNG ideale produrrebbe. Inoltre, poiché esistono un numero infinito di proprietà statistiche da poter testare, non si può certificare la randomicità di un RNG dal suo successo in tutti i test, perché è inevitabile che tale successo si riferisca ai soli test disponibili attualmente e non è implicito che passi anche quelli (magari non ancora implementati) in grado di testare altre grandezze.

3.6 I metodi di post-processing

3.6.1 L'algoritmo di Von Neumann

La procedura di Von Neumann è una procedura estrattiva che permette di ridurre il bias di sequenze di bit casuali indipendenti, sfruttando la simmetria derivante dalla commutatività del prodotto di due probabilità per estrarre bit casuali genuini. Formalmente, essa è una mappa ψ_1: $\Omega \to \Omega$ definita da

$$\psi_1(x_1, \ldots, x_{2n+1}) = \psi_1(x_1, \ldots, x_{2n}) = (y_1, \ldots, y_k)$$

dove $y_i = x_{2m_i}$ e $m_i \in \{m_1, \ldots, m_k\}$ è un indice appartenente all'insieme di tutti gli indici tali che $x_{2m} \neq x_{2m-1}$.
L'output y del metodo pertanto è un vettore di bit genuini che 'salva' le coppie diverse di bit di x.
Il funzionamento della procedura può essere visualizzato tramite il lancio di una moneta truccata. Si consideri una moneta tale da dare una testa con una probabilità p e una croce con probabilità q, con $p \neq q$. Associando dei bit ai risultati dei lanci ripetuti della moneta, è possibile utilizzarla come generatore di numeri casuali. Tale RNG però sarà sempre inevitabilmente affetto da bias, in quanto per definizione esso produrrà un numero maggiore di uno che non di zeri. Tuttavia tale difetto può essere ovviato se si effettuano un numero pari di lanci e si associa un uno ogniqualvolta si ottiene una testa seguita da una croce e uno zero quando si ottiene una croce seguita da una testa. Poiché è sempre vero che $pq = qp$, in tal modo è possibile produrre una sequenza di bit casuali genuini. Tale è il metodo di Von Neumann.
Nel seguito verrà mostrata l'importanza dell'ipotesi sull'indipendenza degli elementi della stringa di partenza e come essa infici la randomicità della stringa prodotta.

3.6.2 L'algoritmo iterato di Peres

Un miglioramento dell'algoritmo di Von Neumann venne proposto da Peres in [4] nel 1992, il quale definì una procedura ricorsiva da applicare alla sequenza di bit casuali indipendenti affetta da bias per estrarre un numero maggiore di bit casuali genuini rispetto al metodo precedente. Formalmente, l'algoritmo di Peres è una procedura estrattiva iterativa ψ_ν: $\Omega \to \Omega$ (con $\nu \geq 2$) definita da

$$\psi_\nu(x_1, \ldots, x_{2n}) = \psi_1(x_1, \ldots, x_{2n}) \star \psi_{\nu-1}(u_1, \ldots, u_n) \star \psi_{\nu-1}(v_1, \ldots, v_{n-k})$$

dove la stringa $x = (x_1, \ldots, x_{2n})$ rappresenta la sequenza iniziale di tutti i bit da processare, il vettore $u = (u_1, \ldots, u_n)$ è costituito dalla somma in modulo a 2 di x (ovvero $u_i = x_{2i-1} \oplus x_{2i}$), $v = (v_1, \ldots, v_{n-k})$ è il vettore che contiene nell'ordine tutti i bit che si ripetono di x, la ψ_1 è la procedura estrattiva di Von Neumann descritta nella sezione precedente e $\star$ rappresenta l'operatore di concatenazione. L'azione della procedura su stringhe di lunghezza dispari si definisce in egual modo.
In particolare egli dimostrò la seguente convergenza uniforme

$$\lim_{\nu \to \infty} r_\nu(p) = h(p)$$

Il significato di tale risultato è che il rate di estrazione della procedura di Peres tende all'entropia per un numero infinito di iterazioni, ovvero che essa può in linea teorica estrarre il massimo numero di bit genuini possibile dalla stringa iniziale.
La sua elevata efficienza deriva da un utilizzo di ulteriori simmetrie rispetto al metodo di Von Neumann, che può essere intuito tramite un esempio che sfrutta nuovamente i lanci ripetuti di monete truccate. Se ne considerino a tale scopo quattro lanci consecutivi e si indichi con T un esito corrispondente ad una testa e con C l'esito complementare. Il metodo di Von Neumann non estrae alcun bit se essi danno coppie di risultati uguali ($CCTT$, ad esempio). Tuttavia, se si osserva che le due possibilità $CCTT$ e $TTCC$ sono equiprobabili e si decide di aggiungere uno zero nella stringa prodotta dalla procedura di Von Neumann ogniqualvolta si verifica la prima condizione e un uno nella seconda, è evidente che così facendo si ottengono nuovi bit casuali genuini dalla stringa iniziale. Vi è però una simmetria aggiuntiva che permette di esaurire i possibili pattern di quattro lanci consecutivi che si possono verificare, producendo un bit a seconda se la coppia T-C o C-T è stata ottenuta nella prima o nella seconda coppia di lanci. Poiché le sequenze $CCTC$, $CCCT$, $CTCC$, $TCCC$ (e quelle che si ottengono scambiando T con C) sono tutte equiprobabili, il bit estratto sarà genuino.
Ricapitolando, il metodo iterato di Peres è costituito dai seguenti punti:

- Si costruisce una nuova sequenza di bit u a partire da quella iniziale x, calcolando la somma in modulo a 2 di quest'ultima, ovvero: $u_i = x_{2i-1} \oplus x_{2i}$
- Si costruisce un'ulteriore sequenza di bit v, che tiene conto delle coppie di bit uguali presi nella stringa x: $v_j = x_{2i_j}$ dove l'indice $i_j \in \{i_1, \ldots, i_{n-k}\}$ è tale che $x_{2i_j} = x_{2i_j-1}$
- Si applica il metodo di Von Neumann alla intera stringa x, producendo una sequenza di bit derivanti dalle k coppie diverse
- Si itera il processo applicandolo alle nuove stringhe u e v, finché esse contengono almeno due elementi
- Si concatenano le stringhe prodotte dall'algoritmo

In appendice è possibile trovare l'implementazione utilizzata di tale metodo.

3.6.3 Il metodo delle differenze temporali

Il metodo delle differenze temporali, a differenza degli algoritmi visti precedentemente, è una regola che permette di generare bit casuali genuini a partire da misurazioni degli intervalli temporali che intercorrono tra le rivelazioni di fenomeni fisici poissoniani, sfruttando il fatto che per definizione gli eventi poissoniani siano indipendenti e che gli intervalli di tempo

frapposti tra due rivelazioni consecutive seguano la densità di probabilità esponenziale $\lambda e^{-\lambda t}$. Se si considera infatti una coppia di intervalli temporali (t_1, t_2) individuati da tre rivelazioni consecutive di un campione, per l'indipendenza degli eventi è impossibile che la condizione $t_1 > t_2$ avvenga più frequentemente di quella opposta $t_1 < t_2$. Tale equiprobabilità può essere utilizzata (i.e. [5]) per generare dei bit casuali, ponendo che ogniqualvolta si verifichi $t_1 > t_2$ venga generato uno zero e un uno altrimenti.
Poiché le misurazioni degli intervalli temporali nei time of arrival QRNG viene ottenuta tramite una discretizzazione del tempo, gli intervalli t_1 e t_2 verranno sempre sostituiti da due interi n_1 e n_2. Ciò determina una maggiore probabilità del caso p($n_1 = n_2$) rispetto al caso p($t_1 = t_2$) e richiede una modifica della versione basilare del metodo sovraesposto che prevede lo scarto dei dati che producono la condizione di uguaglianza per non andare ad inficiare le proprietà statistiche delle stringhe di bit prodotte.

3.6.4 Il metodo pari-dispari

Il metodo pari-dispari è una regola che fornisce bit casuali genuini sfruttando l'asimmetria della distribuzione di probabilità poissoniana. Infatti, se si considera una rivelazione di un fotone come un evento poissoniano con rate medio $\mu = \lambda T$, si può calcolare la probabilità di ottenere rispettivamente un numero pari e dispari di rivelazioni tramite le

$$P(2n) = \sum_{n=0}^{\infty} \frac{(\lambda T)^{2n}}{(2n)!} e^{-\lambda T} = \frac{1 + e^{-2\lambda T}}{2} \qquad P(2n+1) = \sum_{n=0}^{\infty} \frac{(\lambda T)^{2n+1}}{(2n+1)!} e^{-\lambda T} = \frac{1 - e^{-2\lambda T}}{2}$$

Tali funzioni al crescere del rate medio μ tendono ad uno stesso valore (dal grafico riportato in figura (4.1) è possibile vedere come esse siano praticamente identiche già da μ pari a 10). Fissato λ, ciò comporta l'esistenza di un valore di T tale da rendere praticamente equiprobabili le due eventualità. Il metodo consiste pertanto nell'individuare un intervallo di tempo in cui vengono registrate in media una decina di eventi e nel creare bit casuali contando le rivelazioni avvenute in quel tempo di campionamento, producendo un 1 se esso è dispari e uno 0 altrimenti (i.e. [6]).

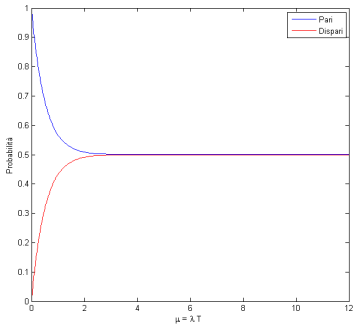


Figura 3.1: Grafici delle probabilità di ottenere un numero pari o dispari di eventi in funzione del rate medio μ. Si noti come al crescere di μ le due funzioni tendano ad uno stesso valore.

Capitolo 4

L'analisi dati

4.1 Presentazione dati sperimentali e loro analisi

Per ottenere diversi regimi di luminosità si è situato al di sopra degli SPAD un LED a luminosità variabile tramite un apposito trimmer e si è sigillato l'apparato in una scatola nera in maniera tale da ridurre il più possibile i dark count. Tramite il software in dotazione del LinoSPAD, si è variato il tempo di integrazione e la luminosità del LED per quattro volte, raccogliendo, per ogni finestra temporale scelta, prima un campione di dark count e successivamente un campione con LED acceso e regolato opportunamente per evitare il più possibile la saturazione dei pixel. Si sono scelti diversi tempi di integrazione per poter inferire la dipendenza del risultati dai due principali difetti sperimentali degli SPAD: il deadtime e l'afterpulsing. Per quanto riguarda il primo, andando ad aumentare i tempi di integrazione si è cercato di ridurre progressivamente il numero di fotoni scartati automaticamente. Infatti, fissato un numero di rivelazioni n, il tempo morto totale risulta essere sempre pari a $n\tau_{dt}$ (dove $\tau_{dt} = 40$ ns per il LinoSPAD è il tempo di deadtime singolo) e aumentando la durata della finestra temporale si tende a far diminuire il rapporto

$$\frac{n\,\tau_{dt}}{\text{time window}}$$

riducendo così la probabilità che un evento cada nel tempo morto totale e non venga rilevato. Inoltre la necessaria diminuzione della luminosità del LED per evitare la saturazione di un numero cospicuo di pixel all'aumentare della time window ha permesso di ridurre la probabilità che un evento reale venisse scambiato per un afterpulse nella fase di post-processing. Il numero di cicli di acquisizione è stato sempre scelto in modo tale da ottenere almeno 75 000 rivelazioni per ogni pixel e da avere sempre meno di 512 eventi per frame, cosicché da poterli sempre considerare come indipendenti. I dettagli relativi alle prese dati sono stati riportati nella tabella (4.1).

Time window (μs)	Numero frame	Numero pixel saturati	Dark count			Numero eventi		
			LLP	NLP	MLP	LLP	NLP	MLP
320	7000	0	2940	8571	33 756	286 868	378 523	547 472
480	8000	0	4986	150 235	443 990	142 376	298 090	582 111
600	8000	1	6115	184 330	545 282	228 529	423 943	774 256
1200	5000	5	7288	223 113	//	76 330	28 4510	//

Tabella 4.1: Dettagli relativi alle prese dati suddivisi per tipologia di pixel

A ciascun set di dati sono stati applicati i metodi di post-processing sopra riportati trascritti in script di MATLAB ottimizzati in termini di velocità e uso di risorse.
L'algoritmo di Peres ha richiesto prima del post-processing, la conversione del vettore dei tag

temporali prodotti dal LinoSPAD in un stringa di zeri e uno. Per ciascun frame infatti sono state create delle stringhe di bit tali da avere degli uno in corrispondenza delle rivelazioni ottenute e degli zeri altrimenti. La loro lunghezza complessiva è stata posta pari a

$$\frac{\text{time window}}{\text{time bin}}$$

dove con time bin ci si riferisce alla risoluzione sperimentale pari a 17.86 ps. Essa è variata da un minimo di 18 Mbit ad un massimo di 67 Mbit, rispettivamente in corrispondenza della finestra temporale da 320 μs e da 1.2 ms. Poiché le operazioni su tali stringhe si sono rivelate avere un costo computazionale estremamente elevato (si vedano i tempi di esecuzione della tabella (4.2)), si è deciso di effettuare l'analisi sui pixel maggiormente significativi, selezionando così un sottoinsieme di 31 pixel di particolare interesse dai 256 pixel totali. Per l'analisi è stato utilizzata la workstation HP Z240.
Infine si è provato a stimare l'effetto degli afterpulse sugli output dei diversi metodi. Per farlo si sono rimossi dai tag iniziali tutte le rivelazioni che distavano temporalmente da quelle precedenti di una quantità inferiore a $2\tau_{ap} = 1\,\mu$s, sovrastimando volutamente per sicurezza l'intervallo di tempo entro il quale la probabilità di afterpulsing risulta essere massima. Per comodità di notazione, ci si riferirà a tale operazione con l'acronimo APT (AfterPulse Treatment).

4.2 Confronto risultati del post-processing

4.2.1 Tempi di esecuzione

Si riportano in primis i tempi di esecuzione dei diversi metodi che sono stati misurati tramite l'apposito comando dell'ambiente MATLAB. Per renderli confrontabili, si è diviso il tempo totale impiegato dai diversi algoritmi per il numero di pixel analizzati e il numero di frame del file corrispondente.

Tabella 4.2: Tempi di esecuzione dei metodi di post-processing

Time window (μs)	Peres	Pari-dispari	Diff. di tempo
	Tempo (s)	Tempo (ms)	Tempo (ms)
320	0.62	1.1	1.8
480	1.28	1.2	1.4
600	1.65	1.3	1.9
1200	3.41	1.3	1.3

Si noti come i tempi relativi all'algoritmo di Peres siano risultati più elevati rispetto a quelli degli altri due metodi di diversi ordini di grandezza.

4.2.2 Numero di bit e bias

Si presentano ora in forma tabulare la lunghezza delle stringhe prodotte (in bit), il loro bias e il relativo p-value. Per comodità di lettura si sono suddivisi i pixel riportati in gruppi da 3 a seconda della loro tipologia di appartenenza in termini di luminosità. Dal punto di vista dell'efficienza dei metodi di post-processing utilizzati, si evince dalle tabelle $(4.3 - 4.6)$ che l'algoritmo di Peres è nettamente più prestante della regola pari-dispari e del metodo delle differenze di tempo, in quanto produce output più lunghi fino a 200 volte rispetto al primo e a 40 volte rispetto al secondo. Tuttavia, bisogna altresì notare che in generale l'algoritmo di Peres risolve in maniera deludente il problema del bias, poiché i valori che esso produce risultano in prima analisi sistematicamente superiori a quelli degli altri due metodi e con p-value inammissibili. Nonostante ciò, si può notare come l'APT consenta di

diminuire drasticamente tale difetto, fino a rendere il metodo qualitativamente il migliore (a meno di alcuni p-value ancora critici). Il bias assoluto medio infatti risulta essere sempre il più alto dei tre precedentemente a tale operazione, mentre post-APT risulta essere sempre il più basso (si vedano le tabelle $(4.7 - 4.8)$ per un riscontro numerico).
Va inoltre osservato che tendenzialmente (come si nota dai segni dei bias riportati nella tabella relativa alla finestra temporale più lunga) l'algoritmo di Peres produce un numero maggiore di uno rispetto agli altri metodi, condizione che probabilmente si verifica per via del forte sbilanciamento tra il numero di uno e di zeri della stringa non processata che potrebbe richiedere la generazione di un numero di uno elevato per essere marginata.
Per quanto concerne il metodo pari-dispari e il metodo delle differenze di tempo, il secondo è stato quello che ha dato risultati migliori, sia in termini di bias medio assoluto che di efficienza.
Infine si sono studiate le diverse riduzioni della lunghezza degli output dovute alla diminuzione forzata dell'entropia associata ai dati pre-processati tramite la fase di APT. A parità di afterpulse tolti infatti, il rapporto tra le lunghezze degli output pre e post-APT è risultato sistematicamente più alto nel caso dell'algoritmo di Peres, il che potrebbe voler dire che esso risente di meno di riduzioni dell'entropia iniziale.

Tabella 4.3: Risultati time window da 320 μs pre e post afterpulse treatment

(a) pre-APT

	Peres			Pari-dispari			Differenze di tempo		
Pixel	Numero di bit	Bias (10^{-3})	p-value bias	Numero di bit	Bias (10^{-3})	p-value bias	Numero di bit	Bias (10^{-3})	p-value bias
1	5 108 824	-14	0	28 000	0.71	0.81	146 616	-0.83	0.53
4	5 265 924	0.019	0.93	29 715	1.2	0.67	151 682	0.45	0.73
8	5 229 617	-13	$6.1{\cdot}10^{-14}$	29 650	-0.74	0.80	150 371	1.15	0.37
54	7 678 390	-13	0	45 223	-4.0	0.086	227 953	-0.068	0.95
81	6 628 552	-1.2	$1.6{\cdot}10^{-9}$	38 429	2.6	0.3	194 386	-0.86	0.45
165	5 588 071	-13	$6.1{\cdot}10^{-14}$	31 492	0.032	0.99	161 768	-2.0	0.11
34	8 743 141	0.034	0.84	52 058	-2.5	0.26	262 452	-1.7	0.076
59	9 299 479	-0.19	0.24	55 787	2.3	0.28	280 871	-0.30	0.75
236	7 498 930	-0.82	$5.7{\cdot}10^{-6}$	43 739	1.3	0.59	222 072	-2.0	0.056

(b) post-APT

	Peres			Pari-dispari			Differenze di tempo		
Pixel	Numero di bit	Bias (10^{-3})	p-value bias	Numero di bit	Bias (10^{-3})	p-value bias	Numero di bit	Bias (10^{-3})	p-value bias
1	3 159 664	-1.2	$4.6{\cdot}10^{-5}$	17 313	-0.78	0.84	86 990	-0.68	0.69
4	3 204 983	-0.39	0.16	17 483	-0.030	0.99	88 473	0.070	0.97
8	3 211 232	0.32	0.25	17 487	-4.1	0.28	88 631	0.68	0.68
54	4 068 066	0.079	0.75	22 750	-1.7	0.61	116 071	1.1	0.45
81	3 744 704	-0.54	0.038	20 973	-1.4	0.69	105 492	-0.01	0.99
165	3 319 584	-0.23	0.40	17 500	1.7	0.66	92 037	0.26	0.88
34	4 393 574	-0.34	0.15	24 500	-4.0	0.21	126 962	-0.18	0.90
59	4 490 165	0.16	0.49	25 994	-4.7	0.13	130 388	-1.3	0.35
236	4 048 310	-0.37	0.14	22 748	-3.5	0.30	115 318	0.92	0.53

Tabella 4.4: Risultati time window da 480 μs pre e post afterpulse treatment

(a) pre-APT

Pixel	Peres			Pari-dispari			Differenze di tempo		
	Numero di bit	Bias (10^{-3})	p-value bias	Numero di bit	Bias (10^{-3})	p-value bias	Numero di bit	Bias (10^{-3})	p-value bias
1	2 695 267	-0.55	0.069	13 614	-4.4	0.30	70 641	-0.96	0.61
4	2 771 046	-8.1	$1.3{\cdot}10^{-14}$	13 944	2.9	0.50	72 889	-0.53	0.78
8	2 763 612	-12	0	13 921	2.3	0.59	72 503	-0.90	0.63
54	7 201 461	0.45	0.015	39 967	1.9	0.45	203 423	-1.2	0.28
81	5 402 414	-0.36	0.092	28 728	-0.91	0.76	149 172	-0.50	0.70
165	3 830 258	-11	$-2.2{\cdot}10^{-16}$	19 965	-3.1	0.38	103 279	-0.75	0.63
34	9 257 600	-0.24	0.14	52 000	1.0	0.64	266 475	-0.39	0.69
59	10 100 827	-0.18	0.26	57 959	-0.026	0.99	293 032	-0.33	0.72
236	7 888 581	-0.75	$2.5{\cdot}10^{-5}$	43 982	1.6	0.50	224 058	-2.6	0.014

(b) post-APT

Pixel	Peres			Pari-dispari			Differenze di tempo		
	Numero di bit	Bias (10^{-3})	p-value bias	Numero di bit	Bias (10^{-3})	p-value bias	Numero di bit	Bias (10^{-3})	p-value bias
1	2 124 671	-0.23	0.51	9 992	-9.8	0.050	26 375	1.8	0.40
4	2 159 967	1.4	$2.4{\cdot}10^{-5}$	9 997	-1.5	0.77	27 033	-1.3	0.54
8	2 169 166	0.091	0.79	9 993	2.3	0.65	26 935	2.9	0.18
54	4 738 467	0.23	0.33	25 011	-8.2	0.0097	63 907	2.5	0.077
81	3 858 110	-0.045	0.86	19 989	-5.6	0.11	51 295	-1.5	0.33
165	2 842 575	-0.13	0.66	13 992	4.1	0.34	36 540	-0.59	0.75
34	5 711 103	-0.059	0.78	30 311	1.67	0.56	78 817	1.34	0.28
59	5 955 771	-0.10	0.62	32 000	0.16	0.96	82 663	1.55	0.21
236	5 160 433	-0.067	0.76	27 976	-4.9	0.099	70 587	-0.13	0.92

Tabella 4.5: Risultati time window da 600 μs pre e post afterpulse treatment

(a) pre-APT

Pixel	Peres			Pari-dispari			Differenze di tempo		
	Numero di bit	Bias (10^{-3})	p-value bias	Numero di bit	Bias (10^{-3})	p-value bias	Numero di bit	Bias (10^{-3})	p-value bias
1	4 304 496	0.29	0.24	21 989	-0.21	0.95	115 275	0.89	0.55
4	4 412 301	-0.27	0.25	21 999	-4.0	0.23	118 487	-3.5	0.015
8	4 408 492	-13	$1.2{\cdot}10^{-14}$	21 999	-3.4	0.31	118 189	-4.5	0.0022
54	9 875 364	-13	$1.1{\cdot}10^{-16}$	55 688	2.8	0.18	280 720	-2.0	0.035
81	7 619 783	-11	$5.8{\cdot}10^{-14}$	41 940	-0.64	0.79	212 484	-1.5	0.18
165	5 627 986	0.078	0.71	29 975	-2.0	0.50	153 828	-2.2	0.091
34	1 244 4660	-1.0	$2.1{\cdot}10^{-12}$	71 376	-3.7	0.048	359 915	-3.0	0.00041
59	13 510 679	-16	$5.7{\cdot}10^{-14}$	77 985	-3.0	0.099	393 666	-0.29	0.71
236	10 609 621	-0.18	0.23	59 970	-1.9	0.34	303 061	-2.1	0.021

(b) post-APT

Pixel	Peres			Pari-dispari			Differenze di tempo		
	Numero di bit	Bias (10^{-3})	p-value bias	Numero di bit	Bias (10^{-3})	p-value bias	Numero di bit	Bias (10^{-3})	p-value bias
1	3 283 652	-0.20	0.47	15 995	4.1	0.30	83 945	-0.35	0.84
4	3 340 264	0.049	0.86	16 000	-9.3	0.019	85 550	-1.4	0.41
8	3 357 361	-0.54	0.046	16 000	-4.6	0.25	86 033	-2.2	0.20
54	6 338 641	0.42	0.032	33 999	1.6	0.55	173 197	-0.57	0.64
81	5 302 627	-0.79	$2.7{\cdot}10^{-4}$	27 991	2.2	0.46	142 064	0.25	0.85
165	4 063 712	-0.17	0.49	20 000	-1.8	0.61	106 086	-0.51	0.74
34	7 496 479	-0.41	0.024	41 402	4.0	0.10	208 739	0.28	0.80
59	7 771 045	-0.91	0	42 000	0.38	0.88	217 593	0.56	0.60
236	7 945 378	0.0076	0.97	36 000	-1.5	0.56	186 272	0.44	0.71

Tabella 4.6: Risultati time window da 1200 μs pre e post afterpulse treatment

(a) pre-APT

Pixel	Peres			Pari-dispari			Differenze di tempo		
	Numero di bit	Bias (10^{-3})	p-value bias	Numero di bit	Bias (10^{-3})	p-value bias	Numero di bit	Bias (10^{-3})	p-value bias
1	1 463 137	-12	$5.9{\cdot}10^{-14}$	6 250	11	0.090	36 353	0.95	0.72
4	1 492 483	-0.76	$6.5{\cdot}10^{-2}$	6 882	1.5	0.81	37 216	-2.4	0.36
8	1 488 584	-12	$5.1{\cdot}10^{-14}$	6 727	2.5	0.69	37 066	-3.9	0.13
54	8 618 515	-12.4	0	47 312	1.2	0.60	237 964	-0.98	0.34
81	5 802 088	-1.3	$1.2{\cdot}10^{-9}$	30 708	3.5	0.21	155 959	-1.1	0.38
165	3 457 260	-11.5	0	17 491	-6.9	0.068	90 270	-0.52	0.75

(b) post-APT

Pixel	Peres			Pari-dispari			Differenze di tempo		
	Numero di bit	Bias (10^{-3})	p-value bias	Numero di bit	Bias (10^{-3})	p-value bias	Numero di bit	Bias (10^{-3})	p-value bias
1	1 242 036	-3.0	$1.3{\cdot}10^{-11}$	5 000	7.4	0.30	29 237	-2.1	0.47
4	1 259 728	-0.63	0.16	5 000	5.6	0.43	29 682	1.9	0.52
8	1 258 440	-0.55	0.22	5 000	6.6	0.35	29 657	0.83	0.78
54	6 071 691	-0.029	0.89	32 003	1.1	0.70	161 680	-1.3	0.29
81	4 489 407	-0.85	$3.2{\cdot}10^{-4}$	22 500	-2.5	0.45	58 257	0.090	0.95
165	2 743 213	-1.0	$5.7{\cdot}10^{-4}$	13 496	0.67	0.88	68 499	-0.71	0.71

Tabella 4.7: Bias assoluti medi dei diversi output

(a) pre-APT

Time window (μs)	Peres	Pari-dispari	Diff. di tempo
	Bias assoluto medio (10^{-3})	Bias assoluto medio (10^{-3})	Bias assoluto medio (10^{-3})
320	4 ± 1	1.5 ± 0.2	1.2 ± 0.2
480	3.6 ± 0.9	3.2 ± 0.5	1.4 ± 0.2
600	3.4 ± 1	2.5 ± 0.3	1.7 ± 0.2
1200	5 ± 1	3.2 ± 0.6	1.8 ± 0.3

(b) post-APT

Time window (μs)	Peres	Pari-dispari	Diff. di tempo
	Bias assoluto medio (10^{-4})	Bias assoluto medio (10^{-3})	Bias assoluto medio (10^{-3})
320	3.7 ± 0.7	2.9 ± 0.4	1 ± 0.1
480	5 ± 1	3.4 ± 0.5	1.4 ± 0.2
600	4.1 ± 0.9	2.2 ± 0.3	0.9 ± 0.1
1200	7.7 ± 2	2.7 ± 0.5	1.6 ± 0.3

4.2.3 Autocorrelazione

Si mostrano ora i grafici dei coefficienti seriali di autocorrelazione delle stringhe prodotte dai diversi metodi per lag compresi tra 1 e 1000. Per aumentare la loro fruibilità, si sono prodotte due rette orizzontali per avere una rappresentazione grafica della soglia di accettabilità dei risultati. Più precisamente, poiché si dimostra che per ogni lag k i valori di a_k possono essere messi in corrispondenza uno ad uno con una variabile aleatoria con distribuzione gaussiana di varianza 1 e media 0, le rette orizzontali rappresentano i valori di a_k corrispondenti ad un p-value pari al livello di significatività $\alpha = 0.01$. Ciò permette visivamente di distinguere tra i valori accettabili di a_k, riprodotti da un RNG ideale il 99% delle volte, e i valori invece problematici che distano più di $2.57\,\sigma$ dal centroide della gaussiana. In tal modo è possibile determinare eventuali andamenti anomali che non possono essere spiegati dai 10 punti (la centesima parte dei 1000 valori di a_k prodotti) aspettati in media al di fuori delle rette di accettabilità.
Tra tutti i grafici riportati nelle figure $(4.1 - 4.2)$, sono risultati di particolare interesse quelle relativi all'algoritmo di Peres. Confrontando questi ultimi infatti, si è notato chiaramente un netto miglioramento dei coefficienti di correlazione a corto raggio (per $k < 200$) post-APT, giacché essi son tornati completamente all'interno della banda di accettabilità. Nel caso (a) invece si son presentati due andamenti tipici prodotti dai pixel in una maniera aleatoria: a seconda dei casi, i coefficienti di autocorrelazione hanno seguito talvolta un andamento decrescente e talaltre un andamento oscillante. Tale effetto potrebbe essere spiegato dalla preponderanza di uno nella stringa di output, che potrebbero essersi distribuiti intervallandosi molte volte con gli zeri (ad esempio andando a produrre frequentemente il byte 1,0,1,0,1,0,1,0). Ciò avrebbe prodotto valori oscillanti di correlazione, perché la stringa traslata ottenuta a partire da quella iniziale avrebbe presentato gli uno in corrispondenza degli uno della stringa originale per lag pari, risultando così correlata, e gli zeri in corrispondenza degli uno nel caso complementare (da cui l'anti-correlazione e il cambio di segno). L'andamento decrescente invece potrebbe essere spiegato nello stesso modo da una maggiore preponderanza di byte con ripetizioni contigue di bit all'interno di essi.
Per completezza, si sottolinea che il trattamento degli afterpulse applicato all'algoritmo di Peres ha migliorato i coefficienti di autocorrelazione, ma altresì non è stato in grado di recuperare tutti gli a_k del caso (a), lasciando al di fuori delle soglie qualche punto per correlazioni molto piccolo (i.e. $k < 5$). Per tale motivo si è ripetuta l'analisi per alcuni pixel dopo aver eliminato tutti gli eventi di afterpulse rilevati in un tempo pari a τ_{Ap} e a $3\tau_{Ap}$. Come si può vedere dai grafici (4.3-4.4), tale rimedio ha ridotto il coefficiente di autocorrelazione problematico oppure l'ha lasciato invariato.
Per quanto concerne gli altri due metodi di post-processing, in tutti i casi analizzati e ivi riportati, i grafici delle autocorrelazioni si sono rivelati sempre accettabili, in quanto i rispettivi plot hanno sempre presentato il numero aspettato di punti al di fuori delle rette. Nonostante ciò, è stato possibile notare come in generale i due metodi abbiano prodotto coefficienti di autocorrelazioni più alti in modulo rispetto all'algoritmo di Peres post-APT. In effetti, si è trovato (si vedano le tabelle $(4.9 - 4.10)$ che in tutti i casi sia i coefficienti medi di correlazione a corto raggio $(k < 5)$ che quelli medi a lungo raggio $(k > 500)$ sono risultati più piccoli per l'ultima versione dell'algoritmo di Peres, il che dimostra quantitativamente la bontà di tale metodo rispetto agli altri.

Tabella 4.8: Coefficienti di autocorrelazione assoluti medi a corto raggio dei diversi output

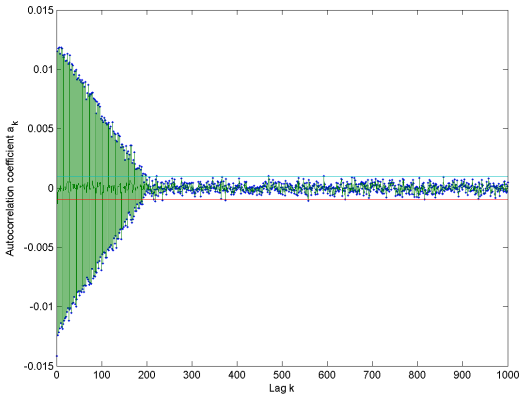
(a) pre-APT

	Peres	Pari-dispari	Diff. di tempo
Time window (μs)	SRC ass. medio (10^{-4})	SRC ass. medio (10^{-3})	SRC ass. medio (10^{-3})
320	6 ± 0.6	4.0 ± 0.3	2.1 ± 0.2
480	5.6 ± 0.6	4.9 ± 0.4	2.9 ± 0.1
600	5.1 ± 0.7	4.2 ± 0.3	2.1 ± 0.1
1200	3.8 ± 0.6	5.3 ± 0.6	2.8 ± 0.3

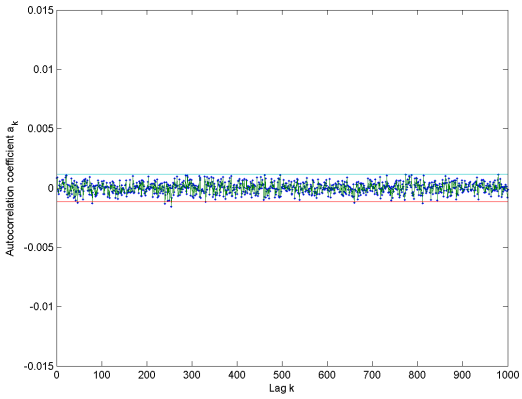
(b) post-APT

	Peres	Pari-dispari	Diff. di tempo
Time window (μs)	SRC ass. medio (10^{-4})	SRC ass. medio (10^{-3})	SRC ass. medio (10^{-3})
320	8.7 ± 0.7	5.1 ± 0.3	2.6 ± 0.2
480	9.8 ± 0.8	6.0 ± 0.5	2.7 ± 0.2
600	8.3 ± 0.7	5.0 ± 0.4	2.0 ± 0.2
1200	9 ± 1	6.5 ± 0.7	2.6 ± 0.3

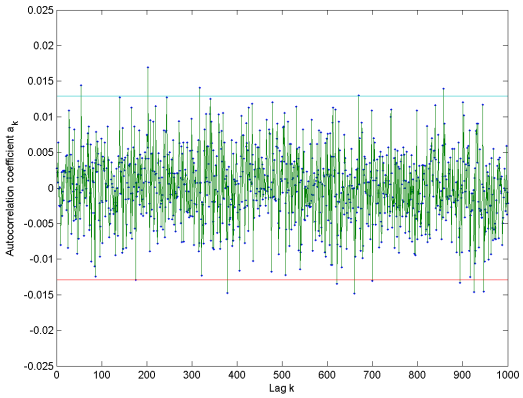
(a) Debiasing pre-APT



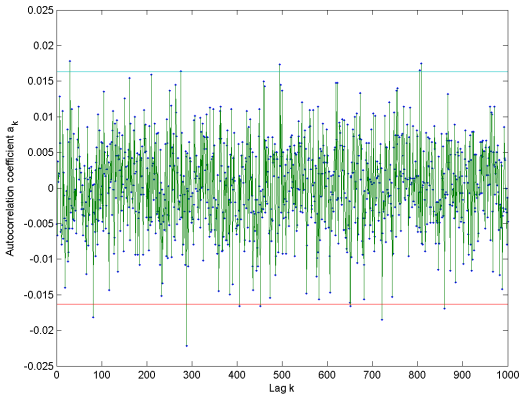
(b) Debiasing post-APT



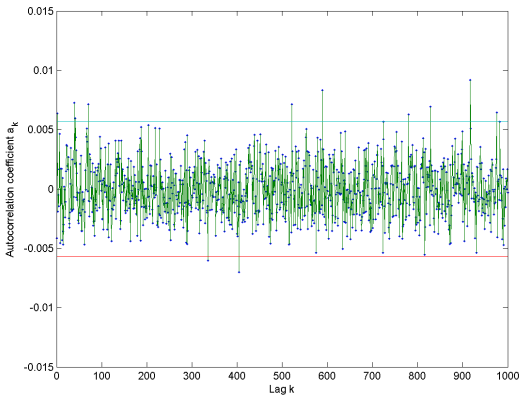
(c) Pari-dispari pre-APT



(d) Pari-dispari post-APT



(e) Differenze di tempo pre-APT



(f) Differenze di tempo post-APT

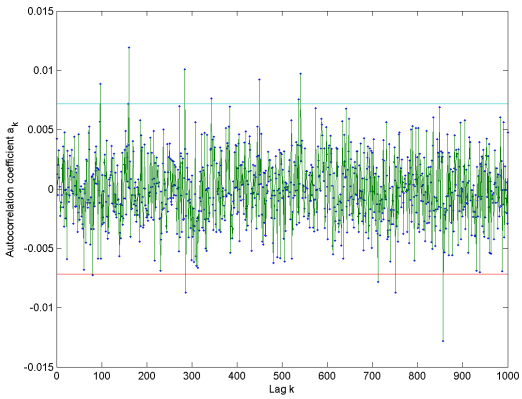
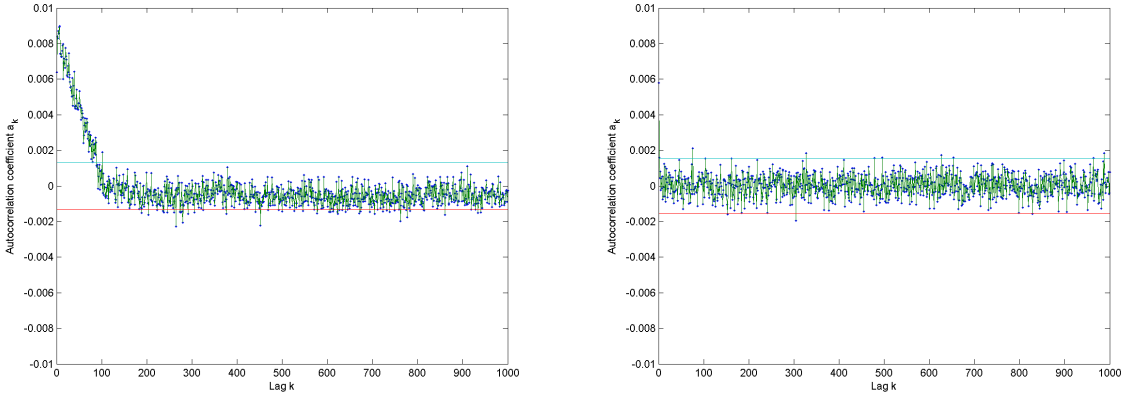


Figura 4.1: Grafici delle autocorrelazioni del pixel 54 per la time window da 480 μs in cui si affiancano gli andamenti di tale variabile per i diversi output ottenuti. Si noti l'andamento "oscillante" dei diversi a_k fino a $k < 200$ nel caso del Peres pre-APT, totalmente recuperato dalla fase di APT. Si osservi inoltre come differenze di tempo e pari-dispari risultino accettabili in tutti i casi riportati.

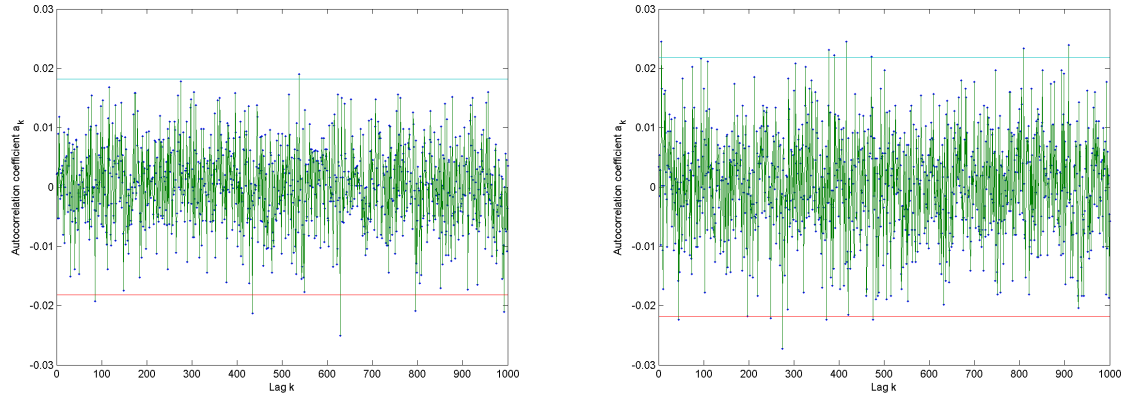
(a) Debiasing pre-APT

(b) Debiasing post-APT



(c) Pari-dispari pre-APT

(d) Pari-dispari post-APT



(e) Differenze di tempo pre-APT

(f) Differenze di tempo post-APT

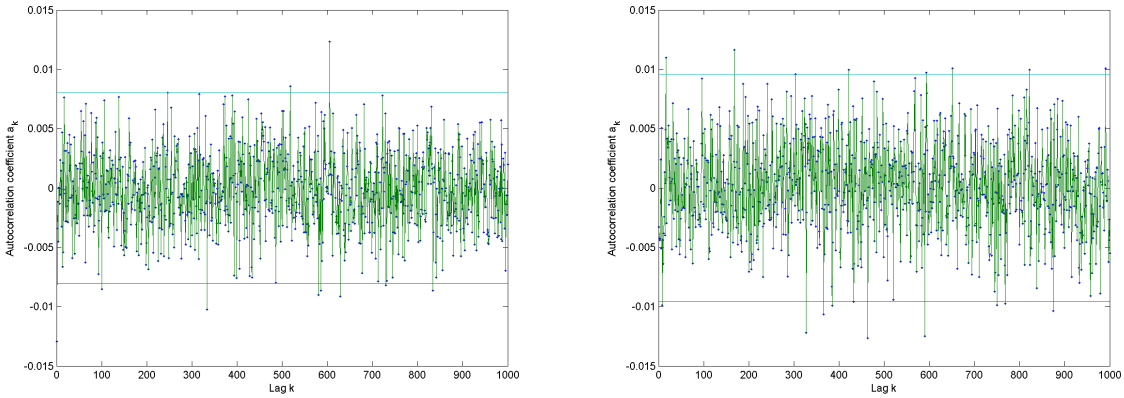


Figura 4.2: Grafici delle autocorrelazioni del pixel 165 per la time window da 480 μs in cui si affiancano gli andamenti di tale variabile per i diversi output ottenuti. Si noti l'andamento "esponenziale" dei diversi a_k fino a $k < 200$ nel caso del Peres pre-APT, totalmente recuperato dalla fase di APT. Si osservi inoltre come differenze di tempo e pari-dispari risultino accettabili in tutti i casi riportati.

(a) Debiasing pre-APT (b) Debiasing post-APT 1 (c) Debiasing post-APT 3

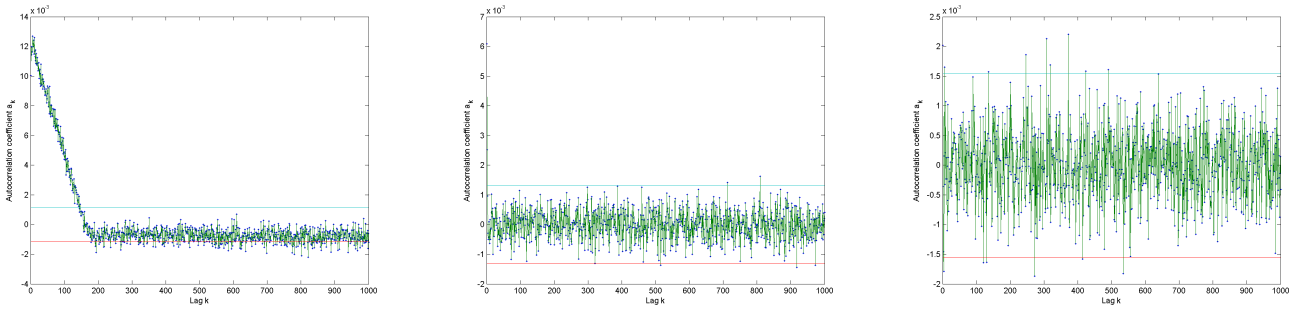


Figura 4.3: Grafico autocorrelazioni del pixel 8 per la time window da 320 μs. Si noti come all'aumentare del tempo minimo richiesto per l'accettabilità degli eventi i primi coefficienti rientrino nella banda individuata dalle rette.

(a) Debiasing pre-APT (b) Debiasing post-APT 1 (c) Debiasing post-APT 3

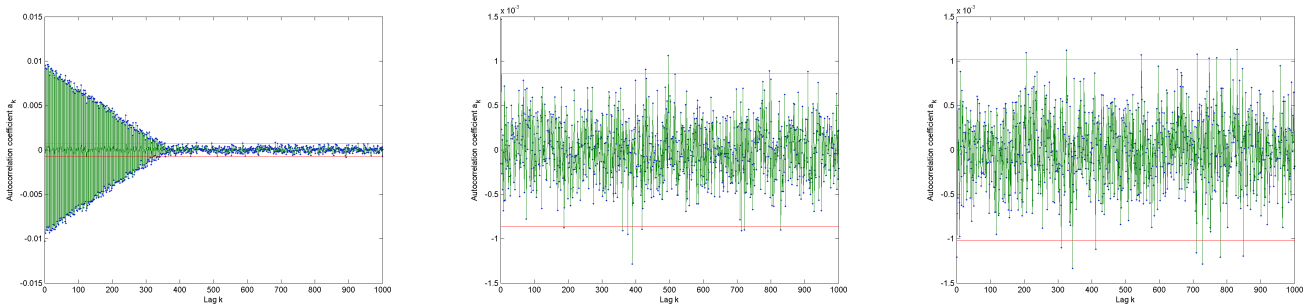


Figura 4.4: Grafico autocorrelazioni del pixel 34 per la time window da 600 μs. Si noti come non vi sia un sostanziale miglioramento tra il caso (b) e il caso (c).

Tabella 4.9: Coefficienti di autocorrelazione assoluti medi a lungo raggio dei diversi output

(a) pre-APT

	Peres	Pari-dispari	Diff. di tempo
Time window (μs)	SRC ass. medio (10^{-4})	SRC ass. medio (10^{-3})	SRC ass. medio (10^{-3})
320	4.5 ± 0.4	4.20 ± 0.08	1.9 ± 0.04
480	4.5 ± 0.3	5.2 ± 0.2	2.3 ± 0.1
600	3.7 ± 0.4	4.2 ± 0.2	1.8 ± 0.09
1200	4.4 ± 0.4	5.5 ± 0.6	2.3 ± 0.3

(b) post-APT

	Peres	Pari-dispari	Diff. di tempo
Time window (μs)	SRC ass. medio (10^{-4})	SRC ass. medio (10^{-3})	SRC ass. medio (10^{-3})
320	3.9 ± 0.2	5.05 ± 0.09	2.53 ± 0.04
480	4.4 ± 0.2	6.3 ± 0.3	2.8 ± 0.1
600	3.4 ± 0.2	5.0 ± 0.2	2.2 ± 0.1
1200	4.2 ± 0.4	6.3 ± 0.7	2.7 ± 0.3

4.2.4 Test AIS31

Come accertamento finale, le stringhe sufficientemente lunghe prodotte dai diversi metodi (ovvero da almeno 5Mbit, condizione verificata soltanto dagli output dell'algoritmo di Peres), sono state sottoposte ai test statistici della suite AIS31 dell'Ufficio Federale per la Sicurezza Informatica tedesco, utilizzata per convalidare il QRNG Quantis della ID Quantique -il più famoso QRNG in commercio.
La suite AIS31 è composta da 8 test riguardanti la ricerca di pattern, la distribuzione delle lunghezze delle run (termine tecnico che sta ad indicare ripetizioni contigue di uno stesso bit all'interno di una stringa), del bias e delle autocorrelazioni delle stringhe su cui viene fatta lavorare. Applicandoli si è trovato è che le stringhe prodotte dal metodo di Peres pre-APT hanno superato i test della suite il 40% delle volte, mentre quelle prodotte post-APT li hanno passati il 100% delle volte.

Capitolo 5

Conclusioni

Dalla analisi svolta è possibile evincere che un post-processing accurato è in grado di dare risultati qualitativamente migliori rispetto alle semplici regole per la generazione di numeri casuali. Come si è più volte sottolineato, l'algoritmo di Peres post-APT presenta ciononostante alcuni difetti che andrebbero risolti prima di una sua effettiva applicazione. Per farlo si potrebbe modificare la fase di APT in maniera tale da rimuovere solo gli uno corrispondenti agli afterpulse senza i corrispondenti zeri oppure aumentare ulteriormente l'intervallo minimo che deve intercorrere tra due eventi per poterli considerare accettabili. Per ovviare invece al tempo di esecuzione particolarmente elevato, si suggerisce di trasportare il codice in un linguaggio di programmazione più veloce come il C++.
Inaspettatamente non è stato riscontrata alcuna dipendenza sostanziale dalle lunghezze delle finestre temporali di integrazione. Ciò potrebbe essere stato dovuto alla contemporanea diminuzione di luminosità imposta dalla saturazione dei LED. Per comprendere maggiormente tale proprietà potrebbe essere utile ripetere l'analisi lasciando invariata la luminosità, focalizzandosi sui soli pixel che non saturano mai.
Infine si sottolinea come il metodo delle differenze di tempo sia in generale più efficace e qualitativamente migliore del metodo pari-dispari. In mancanza della possibilità di una fase di post-processing dedicata, risulta essere pertanto conveniente l'utilizzo del primo, sia per ottenere soluzioni con rate decine di volte più elevati rispetto a quelle basate sul metodo pari-dispari che per avere maggiore robustezza a fronte del fenomeno dell'afterpulsing rispetto all'algoritmo di Peres.
Per concludere, si ringrazia il prof E. Charbon per il prestito del dispositivo LinoSPAD.

Appendice A

Implementazione dell'algoritmo di Peres

Si riporta il codice utilizzato per l'implementazione secondo l'approccio top-down dell'algoritmo di Peres. Il trattamento degli afterpulse è riportato nella costruzione delle stringhe pre-processate.

Listing A.1: main

```
nameOutput = 'results.mat';

DIRUTILS = '../UTILS';
run( [DIRUTILS '/constants.m'] );
run( [DIRUTILS '/selectData.m'] );

for m = 1:nTotalFiles
    %% Lettura informazioni presa dati
    frames = file{2,m}/4;
    f = char( file{1,m} );
    tw = file{3,m};
    nFile = file{4,m};

    lsF = floor( tw/timebin ); %lunghezza sottostringa frame

    disp( f );
    nFramesChosen = 15;

    results = zeros( nPixelsChosen, 8 );
    resultsPerFrame = zeros( frames, 3*nPixelsChosen );
    resultsPerFrame2 = zeros( frames, 2*nPixelsChosen );

    DIROUT = strcat( '../RISULTATI/', DIRNAME,'/', f, '/', 'Debiasing' );
    DIROUT2 = strcat( DIROUT, '/Post-processing/No_dt' );
    DIRDATA = strcat( '../DATI/', DIRNAME);

    for p = startPixel:startPixel+nPixelToElaborate-1
        %% Lettura dati

        n = pixelsChosen( p );
        fprintf('\nNumero pixel: %d\n', n);
        load([DIRDATA '/' f '.mat'],'data');

        nZerosTot = 0;
        nOnesTot = 0;

        tags = zeros( frames*buffer,1 );
        pos = zeros( frames, 1 );
        for k=1:frames
            cf = (k-1)*buffer*pixels;
            cp = (n-1)*buffer;
            a = (k-1)*buffer+1;
            b = k*buffer;
            tags( a:b ) = data( cp+cf+1:cp+cf+buffer );
            pos( k ) = nnz( tags( a:b ) );
        end
        tags( tags >= 2^31 ) = tags( tags >= 2^31 ) - 2^31;
        clear data; %libera la RAM

        if max(pos) == 512
            continue;
        end

        %% Post-processing
        bitsPixel = [];
        bitsPixelAp = [];
        for k=1: frames
            fprintf('\nNumero frame: %d\n', k);
            [bits, bitsCut, bitsFinal] = stringConstruction( tags, k, lsF, n, nFile);

            [ x,y  ] = analysisRNG( bits, 0 );
            [ xC,yC  ] = analysisRNG( bitsCut, 0 );

            nOnesTot = xC + nOnesTot;
            nZerosTot = yC + nZerosTot;

            [ xF,yF  ] = analysisRNG( bitsFinal, 0 );

            resultsPerFrame( k, (p-1)*6+1:p*6 ) = [x,y, xC,yC,xF, yF];
```

```
70
71              bits_d = peres( bitsCut );
72              bitsPixel = [bitsPixel;bits_d];
73
74              nameFile = strcat( 'stringaPixel', num2str( n ),'Frame', num2str( k ), '.bin' );
75              fileID = fopen( [ DIROUT2 '/' nameFile ],'w');
76              fwrite(fileID, bits_d, 'ubit1');
77              fclose(fileID);
78
79          end
80
81          %% Salvataggio risultati
82          nameFile = strcat( 'stringaPixel', num2str( n ), '.bin' );
83          fileID = fopen( [ DIROUT2 '/' nameFile ],'w');
84          fwrite(fileID, bitsPixel, 'ubit1');
85          fclose(fileID);
86
87          [ x, y, bias ] = analysisRNG( bitsPixel, 0 );
88
89          results( p, 1 ) = n; %numero pixel
90          results( p, 2:4 ) = [x,y,bias];
91
92          fprintf('-----------------------------------------------\n');
93
94          save( [DIROUT '/' nameOutput ] );
95      end
96
97
98  end
```

Listing A.2: Peres

```
1   function y = peres( x )
2
3   l = length(x);
4   if mod( l ,2 ) > 0
5       l = l-1;
6   end
7
8   %somma in modulo 2 di x
9   u = mod( x(1:2:l)+x(2:2:l), 2);
10
11  y = x( 2*find( u ) );
12  v = x( 2*find( u == 0 ) );
13
14  if l/2 >= 2 && nnz( u ) > 0
15      z = peres( u );
16      for j = 1: length(z)
17          y(end+1) = z(j);
18      end
19
20  end
21
22  if length( v ) >= 2 && nnz( v ) > 0
23      z = peres( v );
24      for j = 1: length(z)
25          y(end+1) = z(j);
26      end
27  end
28
29  end
```

Listing A.3: Costruzione delle stringhe di input

```
1   %% Costruzione stringa logica totale
2   function [bits, bitsCut, bitsFinal] = stringConstruction( tags, frame, lsF, pixel, file )
3
4   DIR = '../UTILS';
5   run( [DIR '/constants.m'] );
6   load([DIR '/dtPixels.mat'] );
7
8   deadtime = dtPixels( pixel, file );
9   buffer = 512;
10
11  %Determino quanti elementi positivi ci sono in ogni frame del pixel di tags
12  k = frame;
13  a = (k-1)*buffer+1;
14  b = k*buffer;
15  indexMax = find( tags(a:b) >0, 1, 'last');
16  if any( indexMax )
17      pos = indexMax;
18  else
19      pos = 0;
20  end
21
22  %Determino quanti zeri inserire tra gli uno
23  nZeros = zeros( pos+1, 1 );
24  c = 1;
25  d = pos;
26  if d > 0
27      a = (k-1)*buffer;
28      b = a+d;
29      a = a+1;
30
31      Y = diff(tags(a:b));
32      nZeros(c) = tags(a)-1;
33      nZeros(c+1:c+d-1) = Y-1;
34  else
35      nZeros( c ) = 0;
36  end
37
38  nZeros( end ) = lsF - tags( b );
39
40  %definisco un vettore per contenere la stringa totale del frame
41  bits = false( lsF, 1 );
42
43  a = 1; %indice su bits
```

```
44  c = 1; %indice su nZeros
45  if d > 0
46      for i=c:c+d-1
47          a = a+nZeros(i);
48          bits( a ) = 1;
49          a = a+1;
50      end
51  end
52
53  %% Eliminazione deadtime
54  cut_dt =  deadtime;  %numero di elementi della stringa da togliere dopo ogni misura
55  nZerosAp = nZeros;
56
57  c = 2;
58  if d > 0
59      for i=c:length( nZeros )
60              nZeros( i ) = nZeros( i ) - cut_dt;
61      end
62  end
63
64  %definisco un vettore per contenere la stringa generata  con il
65  %nuovo numero di zeri
66  lsFC = sum( nZeros )+ pos;
67  bitsCut = false( lsFC, 1 );
68
69  a = 1; %indice su bits
70  c = 1; %indice su nZeros
71  if d > 0
72      for i=c:c+d-1
73          a = a+nZeros(i);
74          bitsCut( a ) = 1;
75          a = a+1;
76      end
77  end
78
79  %% Eliminazione ap
80
81  %definisco il tempo in cui la probabilit  di rilevare
82  %un afterpulse     massima
83  cut_ap = floor( ap/timebin );
84  cut = cut_dt + 2*cut_ap;
85
86  indicesAp = zeros( length(nZerosAp), 1 );
87  b = 1; %indice su indicesAp
88
89  a = (k-1)*buffer+1;
90  if d > 0
91      for i=a:d+a-1
92          if ismember( i, indicesAp )
93              continue;
94          end
95
96          if i+10 < d+a-1
97              nAp = sum( tags( i+1:i+10 ) <= cut + tags( i ) );
98          else
99              nAp = sum( tags( i+1:a+d-1 ) <= cut + tags( i ) );
100         end
101
102         if nAp > 0
103             indicesAp(b:b+nAp-1) = i+1:i+nAp;
104             b = b+nAp;
105
106             if  tags( i+nAp ) + cut_dt > cut + tags( i )
107                 nZerosAp( i-a+nAp+2 ) = nZerosAp( i-a+nAp+2 ) - cut_dt;
108             else
109                 tmp = sum( nZerosAp( i-a+2:i-a+nAp+1) );
110                 nZerosAp( i-a+nAp+2 ) = nZerosAp( i-a+nAp+2 ) - ( cut - tmp - nAp );
111             end
112
113             pos = pos-nAp;
114         else
115             nZerosAp( i-a+2 ) = nZerosAp( i-a+2 ) - cut_dt;
116         end
117     end
118 end
119
120 indicesAp( indicesAp == 0 ) = [];
121
122 %tolgo tutti gli zeri compresi tra
123 %i veri eventi e gli afterpulse
124 nZerosAp( indicesAp-a+1 ) = [];
125
126
127 lsFrameAp = sum( nZerosAp )+ pos;
128 bitsFinal = false( lsFrameAp, 1 );
129
130 d = pos;
131 a = 1; %indice su bits
132 c = 1; %indice su nZeros
133 if d > 0
134     for i=c:c+d-1
135         a = a+nZerosAp(i);
136         bitsFinal( a ) = 1;
137         a = a+1;
138     end
139 end
140
141 end
```

Bibliografia

[1] M. Stipčević, "Quantum random number generators and their use in cryptography", arXiv.org, 2011.

[2] M. Herrero-Collantes and J.C. Garcia-Escartin, "Quantum random number generators", arXiv.org, 2016.

[3] G. Humer, M. Peev, C. Schaeff, S. Ramelow, M. Stipčević and R. Ursin, "A simple and robust method for estimating afterpulsing in single photon detectors", Journal of Lightwave Technology, 2015.

[4] Y. Peres, "Iterating Von Neumann's procedure for extracting random bits", The Annals of Statistics, 1992.

[5] M. Stipčević and B. M. Rogina, "Quantum random number generators based on photonic emission in semiconductors", Review of Scientific Instruments, 2007.

[6] M. Fürst , H. Weier, S. Nauerth, D.G. Marangon, C. Kurtsiefer and H. Weinfurter "High speed optical quantum random number generation", Optical Society of America, 2010.

[7] W. Killmann and W. Schindler, "A proposal for: functionality classes and Evaluation Methodology for random number generators", referenza tecnica della suite AIS31 disponibile a www.bsi.bund.de/zertifiz/zert/interpr/trngk31e.pdf