EDU-RNG-ASYNC: Educational Asynchronous Random Number Generator PCI Device

Technical Reference Manual

Virtual Hardware Division

September 2026

Contents

0.1 Document Information

Item	Value
Product Name	EDU-RNG-ASYNC
Device ID	0xF00D
Vendor ID	0x1234
Revision	0x10
Interface	Conventional PCI
Generator Type	Pseudo-random
Operation Mode	Asynchronous
Interrupt Support	PCI INTx

1 Overview

The EDU-RNG-ASYNC device is a simple memory-mapped PCI random number generator intended for device driver and operating system education.

Unlike the synchronous version, this device generates random numbers asynchronously. After software requests a number, the device takes a short *generation delay* of a few milliseconds before the result is ready. Completion is indicated through both a status register and a PCI INTx interrupt. This provides a simple example of asynchronous device operation, status polling, and interrupt handling.

2 Features

- Conventional PCI device
- Memory-mapped I/O (MMIO) interface
- 32-bit pseudo-random value generation
- Programmable generator seed
- Asynchronous request-completion model
- Device busy and ready status flags
- PCI INTx completion interrupt

- Write-one-to-clear interrupt acknowledgement
- No DMA or internal request queue

3 PCI Configuration

3.1 Device Identification

Field	Value
Vendor ID	0x1234
Device ID	0xF00D
Revision ID	0x10
Class Code	Other Device
Interrupt Pin	INTA

4 Memory Map

The device exposes a single MMIO Base Address Register (BAR).

4.1 BAR0

Property	Value
BAR Number	0
Size	0x1000 bytes (4 KiB)
Type	Memory-mapped

4.2 BAR0 Layout

Address Range	Description
0x0000-0x0003	RANDOM register
0x0004-0x0007	SEED register
0x0008-0x000B	COMMAND register

Address Range	Description
0x000C-0x000F	STATUS register
0x0010-0x0013	IRQ_STATUS register
0x0014-0x0FFF	Reserved

5 Register Summary

Unless otherwise stated, all registers are 32-bit little-endian values and shall be accessed using aligned 4-byte transactions.

Offset	Register	Size	Access
0x000	RANDOM	32 bits	RO
0x004	SEED	32 bits	WO
0x008	COMMAND	32 bits	WO
0x00C	STATUS	32 bits	RO
0x010	IRQ_STATUS	32 bits	RW

Access legend: RO = Read-only, WO = Write-only, RW = Read-write.

6 Register Descriptions

6.1 RANDOM Register

Offset: 0x000 **Access:** Read-only

Bits	Name	Description
31:0	RANDOM_VALUE	Most recently generated pseudo-random number

Reading this register returns the most recently generated random value. Unlike a synchronous random number generator, reading this register does not initiate generation of a new value.

Example:

```
uint32_t random_value = readl(bar0 + REG_RANDOM);
```

6.1.1 Behaviour

- Returns the most recently generated random value.
- A new value is produced only after a GENERATE command.
- The initial value after device initialisation is zero.
- Reading RANDOM does not clear the READY status.
- Reading RANDOM does not acknowledge an interrupt.
- Reads of sizes other than 32 bits return zero.

6.2 SEED Register

Offset: 0x004 **Access:** Write-only

Bits	Name	Description
31:0	SEED_VALUE	New generator seed

Writing this register seeds the pseudo-random number generator.

Example:

```
writel(12345, bar0 + REG_SEED);
```

6.2.1 Behaviour

- Writing updates the internal generator seed.
- Future generation operations use the new generator state.
- Writing the seed does not initiate random-number generation.
- Writes of sizes other than 32 bits are ignored.

6.3 COMMAND Register

Offset: 0x008 **Access:** Write-only

Bits	Name	Description
0	GENERATE	Start asynchronous random-number generation

Bits	Name	Description
31:1	Reserved	Ignored

Writing a one to the GENERATE bit starts an asynchronous random-number generation operation.

Example:

```
writel(CMD_GENERATE, bar0 + REG_COMMAND);
```

6.3.1 Behaviour

When a generation request is accepted:

1. The READY status bit is cleared.
2. The BUSY status bit is set.
3. A new random value is generated (generation delay, < 10ms).
4. The BUSY status bit is cleared.
5. The READY status bit is set.
6. The completion interrupt status bit is set.
7. The PCI interrupt is asserted.

If GENERATE is written while the device is already busy, the request is ignored. The generation delay is a few milliseconds, and always inferior to 10 ms.

6.4 STATUS Register

Offset: 0x00C **Access:** Read-only

Bits	Name	Description
0	BUSY	Random-number generation operation in progress
1	READY	A generated random value is available
31:2	Reserved	Reads as zero

Example:

```
uint32_t status = readl(bar0 + REG_STATUS);

if (status & STATUS_READY) {
    /* Random value is available */
}
```

6.4.1 BUSY Bit

The BUSY bit is set when a generation operation is in progress.

- `0`: Device is idle.
- `1`: Generation operation in progress.

6.4.2 READY Bit

The READY bit indicates that a generated random value is available.

- `0`: No newly generated value is available.
- `1`: A generated random value is available.

When a new generation operation begins, READY is cleared. When generation completes, READY is set.

6.5 IRQ_STATUS Register

Offset: 0x010 **Access:** Read-write

Bits	Name	Description
0	RANDOM_READY	Random-number generation completed
31:1	Reserved	Reads as zero

This register records pending interrupt causes.

6.5.1 RANDOM_READY Bit

The RANDOM_READY bit is set when an asynchronous generation operation completes. When this bit is set, the device asserts its PCI INTx interrupt. Software acknowledges the interrupt using write-one-to-clear semantics, which means that writing `1` to that bit's location in the IRQ_STATUS register will clear that bit (i.e., set it to `0`).

Example:

```
uint32_t irq_status = readl(bar0 + REG_IRQ_STATUS);

if (irq_status & IRQ_RANDOM_READY) {
    /* Retrieve and process generated random value ... */

    /* Acknowledge the interrupt and clear the corresponding status bit */
    writel(IRQ_RANDOM_READY, bar0 + REG_IRQ_STATUS);
}
```

6.5.2 Behaviour

- Writing a one clears the corresponding interrupt status bit.
- Writing a zero leaves the corresponding bit unchanged.
- When no interrupt causes remain pending, the PCI interrupt is deasserted.
- Reads of sizes other than 32 bits return zero.
- Writes of sizes other than 32 bits are ignored.

7 Programming Model

The device implements an asynchronous request-completion model.

A typical interrupt-driven sequence is:

1. Enumerate the PCI device.
2. Map BAR0.
3. Optionally write a seed to SEED.
4. Enable the device interrupt in the operating system.
5. Write GENERATE to COMMAND.
6. The device enters the BUSY state.
7. After generation completes, the device sets READY in STATUS and RANDOM_READY in IRQ_STATUS.
8. The PCI INTx interrupt is asserted.
9. The interrupt handler reads RANDOM.
10. The interrupt handler acknowledges the interrupt by writing RANDOM_READY to IRQ_STATUS.

8 Interrupt Model

The device uses a conventional PCI INTx interrupt. A generation completion causes the following sequence:

1. RANDOM is updated with the newly generated value.
2. BUSY is cleared.
3. READY is set.
4. IRQ_STATUS.RANDOM_READY is set.
5. The PCI INTx interrupt is asserted.

The interrupt remains asserted while RANDOM_READY remains pending. Software must clear the interrupt cause using a write-one-to-clear operation:

```
writel(IRQ_RANDOM_READY, bar0 + REG_IRQ_STATUS);
```

Once no interrupt causes remain pending, the interrupt is deasserted.

9 Polling Model

The device may also be used without relying on interrupts by polling the STATUS register.

```
writel(CMD_GENERATE, bar0 + REG_COMMAND);

while (!(readl(bar0 + REG_STATUS) & STATUS_READY))
    cpu_relax();

uint32_t random_value = readl(bar0 + REG_RANDOM);
```

Software using polling should still acknowledge any pending interrupt status if interrupts are enabled.

10 Device Behaviour

10.1 Power-Up State

After device initialisation:

- RANDOM contains zero.
- SEED’s value is undefined.
- STATUS.READY is set.
- STATUS.BUSY is clear.
- IRQ_STATUS is zero.
- The PCI interrupt is deasserted.

10.2 Generation in Progress

Only one generation operation may be active at a time. If software attempts to start another generation operation while BUSY is set, the request is ignored. The device does not queue requests.

10.3 Deterministic Sequences

Software may program the SEED register to initialise the pseudo-random number generator to a known state. Reusing the same seed and issuing the same sequence of generation requests allows deterministic sequences to be reproduced.

11 State Transitions

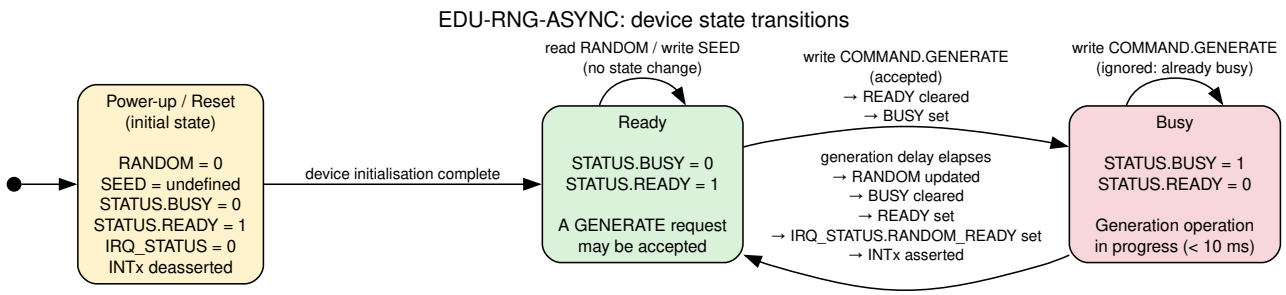

Figure 1: EDU-RNG-ASYNC state transition graph

The device has two primary operational states.

11.1 Ready State

- BUSY = 0
- READY = 1
- A generation request may be accepted.

Writing GENERATE transitions the device to the busy state.

11.2 Busy State

- BUSY = 1
- READY = 0
- A generation operation is pending.
- Additional GENERATE commands are ignored.

After the simulated delay, the device returns to the ready state and generates an interrupt.

12 Appendix A. Constants

```
#define EDU_RNG_VENDOR_ID        0x1234
#define EDU_RNG_DEVICE_ID        0xf00d
#define EDU_RNG_REVISION         0x10

#define EDU_RNG_ASYNC_BAR_SIZE   0x1000
```

13 Appendix B. Register Offsets

```
#define REG_RANDOM      0x000
#define REG_SEED        0x004
#define REG_COMMAND     0x008
#define REG_STATUS      0x00c
#define REG_IRQ_STATUS  0x010
```

14 Appendix C. Bit Definitions

```
#define CMD_GENERATE        (1U << 0)

#define STATUS_BUSY         (1U << 0)
#define STATUS_READY        (1U << 1)

#define IRQ_RANDOM_READY    (1U << 0)
```