Note

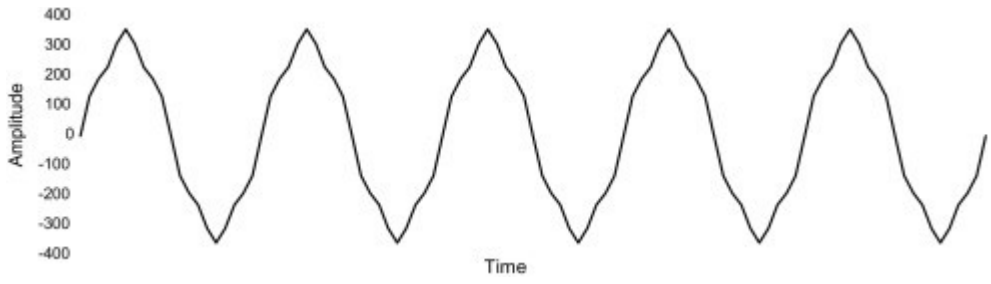
- Expected result: 160 Khz
- CPU clock is 84 MHz (84 million instructions per second), this value is the same for Timer (84 million cycles per second)
- DAC rate

Atmel SAM3X8E SAM3X8C SAM3X4E SAM3X4C SAM3A8C SAM3A8C Datasheet
Atmel-11057-32-bit-Cortex-M3-Microcontroller-SAM3X-SAM3A_Datasheet.pdf

clock
- High precision 8/12 MHz factory trimmed internal RC oscillator with 4 MHz default frequency for fast device startup
- Slow Clock Internal RC oscillator as permanent clock for device clock in low-power mode
- One PLL for device clock and one dedicated PLL for USB 2.0 High Speed Mini Host/Device
- Temperature Sensor
- Up to 17 peripheral DMA (PDC) channels and 6-channel central DMA plus dedicated DMA for High-Speed USB Mini Host/Device and Ethernet MAC

- Low-power Modes
  - Sleep, Wait and Backup modes, down to 2.5 µA in Backup mode with RTC, RTT, and GPBR
- Peripherals
  - USB 2.0 Device/Mini Host: 480 Mbps, 4 Kbyte FIFO, up to 10 bidirectional Endpoints, dedicated DMA
  - Up to 4 USARTs (ISO7816, IrDA®, Flow Control, SPI, Manchester and LIN support) and one UART
  - 2 TWI (I2C compatible), up to 6 SPIs, 1 SSC (I2S), 1 HSMCI (SDIO/SD/MMC) with up to 2 slots
  - 9-channel 32-bit Timer Counter (TC) for capture, compare and PWM mode, Quadrature Decoder Logic and 2-bit Gray Up/Down Counter for Stepper Motor
  - Up to 8-channel 16-bit PWM (PWMC) with Complementary Output, Fault Input, 12-bit Dead Time Generator Counter for Motor Control
  - 32-bit low-power Real-time Timer (RTT) and low-power Real-time Clock (RTC) with calendar and alarm features
  - 256-bit General Purpose Backup Registers (GPBR)
  - 16-channel 12-bit 1 msps ADC with differential input mode and programmable gain stage
  - 2-channel 12-bit 1 msps DAC
  - Ethernet MAC 10/100 (EMAC) with dedicated DMA
  - 2 CAN Controllers with 8 Mailboxes
  - True Random Number Generator (TRNG)
  - Register Write Protection

Example of a distored sin wave



Option 1: Use Timer + ISR

Timer → ISR → DAC

CPU still used in ISR and we still can not reach the maximum frequency as even with the minimized code, which still has many instructions.

The unavoidable overhead includes:

- *Context saving (pushing registers to stack): ~10-20 cycles*
- *Interrupt vector lookup: ~4-6 cycles*
- *ISR prologue/epilogue: ~8-12 cycles*
- *Interrupt return (restoring context): ~10-20 cycles*
- *Interrupt latency (pipeline flush): ~3-5 cycles*

Total overhead: ~35-63 cycles minimum before the actual DAC code even runs.

There is a separate timer (SysTick) running defaultly in Arduino Due and it interrupts our Interrupt. The solution to fix this is: Set our interrupt to higher Priority in **NVIC (Nested Vectored Interrupt Controller)**

Option 2: Use DMA

Memory → DMA (triggered by Timer which the maximum speed is 84 MHz) → DAC

CPU only used in Init

DMA Register	Mapping
`DACC_TPR` (Transmit Pointer Register)	Address of the lookup table
`DACC_CDR` (Conversion Data Register)	DAC data register
`DACC_TCR` (Transmit Counter Register)	Number of samples
`DACC_TNPR` (Transmit Next Pointer Register)	Address of the lookup table
`DACC_TNCR` (Transmit Next Counter Register)	Number of samples

DAC uses a dedicated **Peripheral DMA Controller (PDC)**, not the general-purpose DMAC. The PDC registers are mapped directly into the DACC's memory space.

Result

Number of samples: 8, 15, 30, 60, 120 (lower means higher frequency)