AN13498

Digital Signal Processing for NXP LPC553x/LPC55S3x Using PowerQuad

Rev. 1 — 25 May 2022
Application Note

Contents

1 PowerQuad introduction

Mobile IoT and Context® awareness are growing tremendously. More local digital signal processing is required. Low-power always-on systems are good options for Cortex M33-based MCUs (for leakage reduction and overall low power, considering limited computation).

Arm® Cortex-M33 architecture gears towards energy efficient control applications.

Signal processing lags behind traditional DSP architectures, sometimes as much as 10x-20x in terms of performance due to the following factors:

- Narrow memory width (single 32-bit data bus) – DSPs contain at least two data buses and local memory blocks.
- Limited simultaneous computational capability (for example, one multiplication + add per cycle).
- Not enough registers for intermediate keeping of necessary data.
- No dedicated built-in accelerators for functions, such as, FFT (large load of additions/subtractions) and Biquad Filters.

Although Arm does not bring large-scale DSP improvements to Cortex-M family of cores, it has standardized the DSP library (CMSIS DSP Lib). When using a common standard interface for DSP functions, there is an opportunity to provide a vendor supplied optimizations. User code still uses CMSIS DSP, but NXP can **improve the recipe under the hood**. Accelerating computations cuts power. MCU goes to sleep and then runs slowly at a lower frequency and lower voltage (lowering energy further still). Then, the PowerQuad comes.

The below are mathematical requirements in DSP applications:

- Motion context
  - Matrix operations, Rotation via trigonometric functions, FFT, Filter (FIR/IIR) for calibration.
  - Convolution and correlation for motion feature extraction and matching.
- Voice recognition
  - FFT for spectral analysis, Logarithm, and Mel-Frequency and other windowing (Matrix multiplication), Filter (FIR/IIR), DCT for Cepstrum extraction.
  - Statistical modeling for feature extraction and comparison.
- Neural networks architecture-specific features
  - Matrix MAC
  - Logistic/Sigmoid function (using exponentiation) for perception evaluation (also very useful for statistical distribution analysis.
- Biometrics
  - FFT for Heartbeat monitoring, Arctan/other trig for Fingerprinting.

Now, the PowerQuad can support most mathematical requirements on the hardware. It accumulates the process and saves CPU time for other thread simultaneously.

2 PowerQuad hardware

2.1 PowerQuad computing features

As a hardware module integrated inside the chip, PowerQuad executes the calculation task all on the hardware. It involves various computing engines:

- Transform engine
- Transcendental function engine
- Trigonometry function engine
- Dual biquad IIR filter engine
- Matrix accelerator engine
- FIR filter engine
- CORDIC engine

Table 1 lists the computing features that PowerQuad supports directly.

Table 1. PowerQuad hardware function

Class	Function	Comments
Math	1/x, ln(x), sqrt(x), 1/sqrt(x), e"(x), e"(-x), (x1) / (x2), sin(x), cos(x)	Coprocessor instruction
	arctan(x), arctanh(x)	
Filter	2nd order IIR filter	Coprocessor instruction
	• FIR filter • FIR filter incremental • Correlation • Convolution	
Matrix	• Scale • Addition • Subtraction • Invert • Product • Hadamard product (element-wise product) • Transpose • Dot product	—
Transform	• Complex FFT (complex-valued input sequence) • Real FFT (real-valued input sequence)	—

Table continues on the next page...

Table 1. PowerQuad hardware function (continued)

Class	Function	Comments
	• Inverse FFT • Complex DCT (complex-valued input sequence) • Real DCT (real-valued input sequence) • Inverse DCT	

These functions form the foundation for the implementation of advanced algorithm.

2.2 PowerQuad bus interfaces

PowerQuad is integrated with the Arm Cortex-M33 co-processor Interface. It can be accessed through the co-processor instructions (**MCR** and **MRC**). Also, there are programmable registers designed inside the PowerQuad to connect the AHB bus. User code running on the Cortex-M33 core can read and write its register, as other normal programmable modules. See Figure 1.

However, specific access ways are for the specific usage. Generally, for PowerQuad, Arm Cortex-M co-processor interface, and AHB slave interface are used to deliver the commands/configurations. AHB master interface and the private RAM master interface are used to operate the memory.

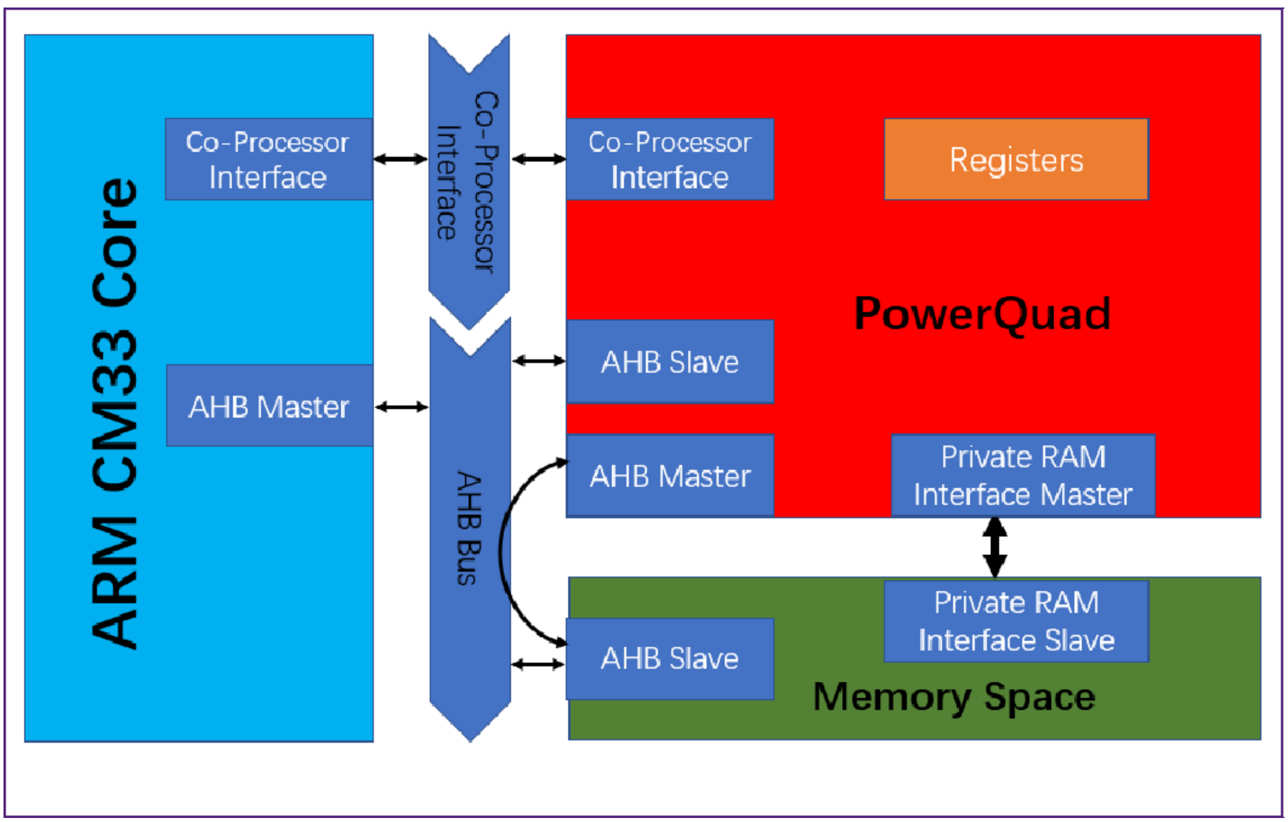

Figure 1. PowerQuad bus interfaces

- Co-processor functions

  When doing the calculation which accepts one number as input parameter and returns one number as output result, they use the Cortex-M Co-processor interface to pass in the input parameter and return the result. For example, the most math functions are implemented in this way. These functions are simple and running very soon.

- Streaming/DMA functions

  When doing the calculation that works on an array of data and the result is another array of data, the PowerQuad uses a DMA-like way to handle the input and output data. Examples of AHB access functions are the transform functions, matrix functions, and most filter functions. When using the PowerQuad for these functions, set some base address registers of PowerQuad, like using DMA. Then, the PowerQuad hardware uses the memory indicated with these addresses automatically when the calculation is launched.

NXP MCUXpresso SDK already provides the driver for PowerQuad. It packs the operations with co-processor interface (co-operator instruments) and AHB bus (functional registers). So, if the users develop their applications with the SDK API, they do not need to care how to select the instructions or register settings.

2.3 PowerQuad memory handlers

When considered as an embedded mathematics computer, the PowerQuad needs numerous data to be processed and produced.

Along with the powerful computing engines, there are four groups for memory handler, which indicate the four memory areas to support the data management requirement of PowerQuad functions.

- Input A. pointer to the input data array 1.
- Input B. pointer to the input data array 2 when necessary. For example, when making the matrix addition, indicate the other matrix by Input B handler.
- Temp. pointer to the temporary memory that keeps the intermediate computational results when necessary (for FFT and Matrix Inversion). Initialize the memory before the current calculation and then clear it later. PowerQuad writes values and reads them automatically during the calculation.

Configure each memory area for the customized format:

- Format of originating data (32-bit fixed, 16-bit fixed or 32-bit float)
- Format of data desired for PowerQuad (float for all except FFT, which is a fixed-point engine)
- Scale of result (PowerQuad can do scaling by power of 2 on the way in its out.)

Users can fill the address of prepared memory into the responding registers in the PowerQuad module, as shown in Table 2.

Table 2. PowerQuad registers for memory handlers

Address	Name	Description	Access	Reset value
`0x000`	OUTBASE	Base address register for output region	RW	0
`0x004`	OUTFORMAT	Data format for output region	RW	0
`0x008`	TMPBASE	Base address register for temp region	RW	0
`0x00C`	TMPFORMAT	Data format for region Temp	RW	0
`0x010`	INABASE	Base address register for input A region	RW	0
`0x014`	INAFORMAT	Data format for region input A	RW	0

Table continues on the next page...

Table 2. PowerQuad registers for memory handlers (continued)

Address	Name	Description	Access	Reset value
`0x018`	INBBASE	Base address register for input B region	RW	0
`0x01C`	INBFORMAT	Data format for region input B	RW	0

PowerQuad can handle the general RAM memory (shared with other AHB masters, like Cortex-M core) and private RAM memory (start from 0xE000_0000, 16 KB). Specially, for private RAM memory, as it is reserved only for PowerQuad, PowerQuad can access it without any arbitration delay, saving time for PowerQuad to get data. Then, PowerQuad can access the private RAM four banks of memory in parallel, giving 128-bit wide. So, it performs some functions even much faster, such as, FFT, FIR, convolution, matrix.

When using the private RAM,

- FFT engine may only use the private memory as temp memory (not as input or output).
- All data in private memory must be floating point. (You can get data in and out of private memory by using the matrix scale operation with private memory being destination).
- The private memory does not provide any scaling. Scaling is only available for data which is being read/written to the system memory.

3 PowerQuad DSP examples

This section describes the basic usage of PowerQaud in application and the PowerQuad APIs during the explanation of demo case.

The demo runs on the LPCXpresso55S36 board with an LCD screen module to show the GUI. In the demo project, a simple framework can switch the separate task as a scheduler. Execute simple tasks one by one, for FFT, matrix, and FIR. With the LCD screen module, the display function is integrated into the framework.

The PowerQuad FFT, matrix, and FIR filter are chosen in this demo. These calculations are popular in most DSP application but usually cost time when implemented by pure software (Arm CMSIS-DSP Lib). PowerQuad vs Arm CMSIS-DSP performance provides a comparison of performance for PowerQuad APIs and Arm CMSIS-DSP API.

This application note does not discuss the details about the calculation process. For further information, see PowerQuad UM and SDK driver code.

A detailed illustration about using PowerQuad APIs is described for FFT cases. The same idea is applied to other cases.

3.1 Hardware environment setup

Before running the DSP example, set up the hardware environment.

- Prepare an LPCXpresso55S36.
- Prepare an LCD module (wave-shape 2.8 inch TFT Shield)
- Connect LCD to LPCXpresso55S36 (J102, J132, J92, J122)
- Connect pin 3 of JP64 to D13 of J9 (As `flexspi` uses pin 1 and 2 of JP64, BK of LCD must jump to D13 of J9.)
- Download image `bin` file located at `.\docs\images` to `flexspi` flash with `blhost.exe`. To download, follow the steps by **NOR FLASH Config, Erase and Program via blhost tool**, as described in LPC553x and LPC55S3x Reference Manuals. Table 3 lists the destination address in external flash for image bins.

Table 3. Image bin file destination address

Image Bin file	Destination address
Wel24b.bin	0x08000000
MAdd24b.bin	0x08030000
MInv24b.bin	0x08060000
MMul24b.bin	0x08090000
Tab24b.bin	0x080C0000

- Download code into internal flash of LPC5536/LPC55S36 device and run.
- To switch between different DSP cases one by one, press the sw3 (USR) button.

3.2 Task schedule with display GUI

To involve the separate cases into one project, implement a scheduler in the demo project. Each case is implemented within a function as the task entry. All the task entries are collected into the task array, `cAppLcdDisplayPageFunc[]`. Also, a hardware thread to capture the button is launched.

Then, the MCU is in the sleep mode until waken up by the key interruption. The key value is changed in the ISR of key interruption. The main loop checks the change of key value and switches to the task with the index (using the key value) in the task list.

```
/* List of lcd display with tasks. */
void (*cAppLcdDisplayPageFunc[])(void) =
   {
      task_pq_fft_128,
      task_pq_fft_256,
      task_pq_fft_512,
      task_pq_mat_add,
      task_pq_mat_inv,
      task_pq_mat_mul,
      task_pq_fir_lowpass,
      task_pq_fir_highpass,
      task_pq_records
    };
    int main(void)
    {
        ...
        while (1)
        {
            keyValue = App_GetUserKeyValue(); /* keyvalue is used as the index of task. */
            if (keyValue != keyValuePre) /* only switch task when keyvalue is changed. */
            {
            App_DeinitUserKey(); /* disable detecting key when changing lcd display. */
              (*cAppLcdDisplayPageFunc[keyValue])(); /* switch to new page with new task. */
              keyValuePre = keyValue;
              App_InitUserKey(); /* enable detecting key for next event. */
            }
             __WFI(); /* sleep when in idle. would wake up when the key interrupt happens caused by
the touch screen. */
          }
      }
```

In each task, it executes the PowerQuad computing to finish a simple task and measure the time for critical operations. Then, it shows the record to the LCD screen module.

3.3 Functions of measuring time

Considering that the functions are usually running fast, interrupt-based timing method is not suitable in the demo case. However, in some test projects specially for measuring, interrupt-based timing method is still available. This method measures plenty times of the target function and gets the average time for one execution.

In this demo, `SysTick` timer is chosen as the timer and the code is portable for the other Arm Cortex-M MCU. Use the 24-bit counter value directly for timing. For the LPC5536/LPC55S36 device which is running at 98 MHz for the clock source of SysTick timer, the maximum timing period is 171 ms.

```
/* Systick Start */
#define TimerCount_Start()    do {                    \
   SysTick->LOAD    =    0xFFFFFF    ;    /*    Set reload register    */\
   SysTick->VAL    =    0    ;         /*    Clear Counter */    \
   SysTick->CTRL    =    0x5 ;         /*    Enable Counting*/    \
} while(0)

/* Systick Stop and retrieve CPU Clocks count */
#define TimerCount_Stop(Value) do {    \
SysTick->CTRL =0; /* Disable Counting */    \ Value = SysTick->VAL;/* Load the SysTick Counter Value
*/ \ Value = 0xFFFFFF - Value;/* Capture Counts in CPU Cycles*/\ } while(0)
```

The usage is:

```
uint32_t calcTime;

TimerCount_Start();
arm_cfft_q31(&instance, gPQFftQ31InOut, 0, 1); /* Calculation. */
TimerCount_Stop(calcTime);

printf("calcTime: %d", calcTime);
```

3.4 FFT demo cases

There are three FFT cases in the demo: 128 points, 256 points, and 512 points. The below lists tips when using PowerQuad FFT engine:

- PowerQuad can support 16/32/64/128/256/512 points for FFT computing engine on the hardware.
- The PowerQuad FFT engine scales the input data by 1/N when computing the FFT (and by extension DCT). If an unscaled result is necessary, multiply the input data (in the INPUT A region) by N manually and scale the inverse FFT scaled by 1/N. It is correct as per the iDFT formula, so no scaling treatment is needed.
- The FFT engine only looks at the bottom 27 bits of the input word, so no pre-scaling can exceed to avoid the saturation.
- The purely real (prefixed by ‘r’ in API name) and the complex flavors of the functions (prefixed by 'c' in API name) expect the input data sequences to be arranged in memory as follows.
- If the sequence x = x0, x1 ... xN-1 are real numbers, then the input array in memory is organized as x[N] = {x0, x1, ... xN-1}.
- If the sequence x = x0, x1 ... xN-1 are complex numbers of the form of (x0_real + i*x0_im), (x1_real + i*x1_im), ... (xN-1_real + i*xN-1_im), then the input array in memory is organized as x[N] = {x0_real, x0_im, x1_real, x1_im, ... xN-1_real, xN-1_im}.
- The output sequence is stored in the memory organized as an array of complex numbers where the imaginary parts are zero for real-valued output data.

When running the PowerQuad Transform engine (include the FFT), only the INPUT A memory handler is used for input and the OUT memory handler is used for output. For the full information about the usage of memory handler for Transform engine, see Table 4.

Table 4. Usage of memory handlers for FFT engine

Operation	Driver function	Access type	Input/ Output data formats	Input A region usage	Input B region	Output region usage	Temp. region usage	Fixed point input/ output scalers	Engine	Uses GPREGs / COMPR EGs?
Complex FFT	`Pq_cfft`	AHB	Fix-16, Fix-32	Input data	N.A.	Output data	N.A.	`Ina_scaler/ Inb_scaler/ Out_scaler`	X_{form}	Yes
Real FFT	`Pq_rfft`	AHB	Fix-16, Fix-32	Input data	N.A.	Output data	N.A.	`Ina_scaler/ Inb_scaler/ Out_scaler`	X_{form}	Yes
Inverse FFT	`Pq_ifft`	AHB	Fix-16, Fix-32	Input data	N.A.	Output data	N.A.	`Ina_scaler/ Inb_scaler/ Out_scaler`	X_{form}	Yes
Complex DCT	`Pq_cdct`	AHB	Fix-16, Fix-32	Input data	N.A.	Output data	N.A.	`Ina_scaler/ Inb_scaler/ Out_scaler`	X_{form}	Yes
Real DCT	`Pq_rdct`	AHB	Fix-16, Fix-32	Input data	N.A.	Output data	N.A.	`Ina_scaler/ Inb_scaler/ Out_scaler`	X_{form}	Yes
Inverse DCT	`Pq_idct`	AHB	Fix-16, Fix-32	Input data	N.A.	Output data	N.A.	`Ina_scaler/ Inb_scaler/ Out_scaler`	X_{form}	Yes

The PowerQuad APIs used in the demo is compatible as the CMSIS-DSP API. CMSIS-DSP users do not need to change the existing codes but can run faster with the implementation of PowerQuad.

Taking FFT of 128 points as examples:

```
extern q31_t    gPQFftQ31In[APP_PQ_FFT_SAMPLE_COUNT_MAX*2u];
extern q31_t    gPQFftQ31Out[APP_PQ_FFT_SAMPLE_COUNT_MAX*2u];
extern q31_t    gPQFftQ31InOut[APP_PQ_FFT_SAMPLE_COUNT_MAX*2u];
extern float32_t gPQFftF32In[APP_PQ_FFT_SAMPLE_COUNT_MAX*2u];
extern float32_t gPQFftF32Out[APP_PQ_FFT_SAMPLE_COUNT_MAX*2u];

void task_pq_fft_128(void)
{
     arm_cfft_instance_q31 instance;
     uint32_t i;
     uint32_t calcTime;

     /* Create the input signal. */
     for (i = 0; i < APP_PQ_FFT_SAMPLE_COUNT_128; i++)
     {
         /* real part. */
         gPQFftF32In[i*2] = 1.5f /* direct current. */
                             + 1.0f * arm_cos_f32( (    2.0f * PI / APP_PQ_FFT_PERIOD_BASE) *
i ) /* low frequence */
                             + 0.5f * arm_cos_f32( (4.0f * 2.0f * PI / APP_PQ_FFT_PERIOD_BASE) *
i ) /* high frequence */
                              ;
            gPQFftF32In[i*2] /= 3.0f; /* make sure the value in (0, 1) */
            /* imaginary part */
            gPQFftF32In[i*2+1] = 0.0f;

         }
    /* PowerQuad FFT can only operate fix-point number. */
         arm_float_to_q31(gPQFftF32In, gPQFftQ31In, APP_PQ_FFT_SAMPLE_COUNT_128*2u);
         for (i = 0u; i < APP_PQ_FFT_SAMPLE_COUNT_128 * 2u; i++)
         {
            gPQFftQ31InOut[i] = gPQFftQ31In[i] >> 5u; /* powerquad fft engine can only accept 27-bit
input data. */
          }

          instance.fftLen = APP_PQ_FFT_SAMPLE_COUNT_128;
          TimerCount_Start(); /* start timing. */
          arm_cfft_q31(&instance, gPQFftQ31InOut, 0, 1); /* computing. */
          TimerCount_Stop(calcTime);

          for (i = 0u; i < APP_PQ_FFT_SAMPLE_COUNT_128 * 2u; i++)
          {
              gPQFftQ31Out[i] = gPQFftQ31InOut[i] « 5u; /* restore the data from 27-bit to 32-bit. */
          }

          arm_q31_to_float(gPQFftQ31Out, gPQFftF32Out, APP_PQ_FFT_SAMPLE_COUNT_128*2u);
arm_cmplx_mag_f32( gPQFftF32Out, gPQFftF32In, APP_PQ_FFT_SAMPLE_COUNT_128);
          /* Todo ...
          * - Record the time.
          * - Display the waveform.
          */
   }
```

`arm_cfft_q31()` calls the PowerQuad driver `PQ_TransformCFFT()`/`PQ_TransformIFFT()`.

```
  void arm_cfft_q31(const arm_cfft_instance_q31 *S, q31_t *p1, uint8_t ifftFlag, uint8_t
bitReverseFlag)
```

```
{
     assert(bitReverseFlag == 1);

     q31_t *pIn = p1;
     q31_t *pOut = p1;
     uint32_t length = S->fftLen;

     PQ_DECLARE_CONFIG;
     PQ_BACKUP_CONFIG;
     PQ_SET_FFT_Q31_CONFIG;

     if (ifftFlag == 1U)
     {
         PQ_TransformIFFT(POWERQUAD_NS, length, pIn, pOut);
      }
      else
      {
          PQ_TransformCFFT(POWERQUAD_NS, length, pIn, pOut);
       }

          PQ_WaitDone(POWERQUAD_NS);

          PQ_RESTORE_CONFIG;
}
```

Then the `PQ_TransformCFFT()` function configures the PowerQuad registers to set the input/output and the length of memory, then launches the computing by enabling the PowerQuad as CFFT engine. After these operations, the PowerQuad can work.

```
void PQ_TransformCFFT(POWERQUAD_Type *base, uint32_t length, void *pData, void *pResult)
{
     assert(pData);
     assert(pResult);

     base->OUTBASE = (int32_t)pResult;
     base->INABASE = (int32_t)pData;
     base->LENGTH = length;
     base->CONTROL = (CP_FFT « 4) | PQ_TRANS_CFFT; /* Launch the computing task. */
}
```

When the computing is done, the `INST_BUSY` is asserted. Users can use the `PQ_WaitDone()` function to wait the PowerQuad done.

```
void PQ_WaitDone(POWERQUAD_Type *base)
{
    /* wait for the completion */
    while ((base->CONTROL & INST_BUSY) == INST_BUSY)
    {
           __WFE(); /* Enter to low power. */
    }
}
```

When running the demo project, there display pages on the LCD screen module for each FFT demo case are as shown in Figure 2.

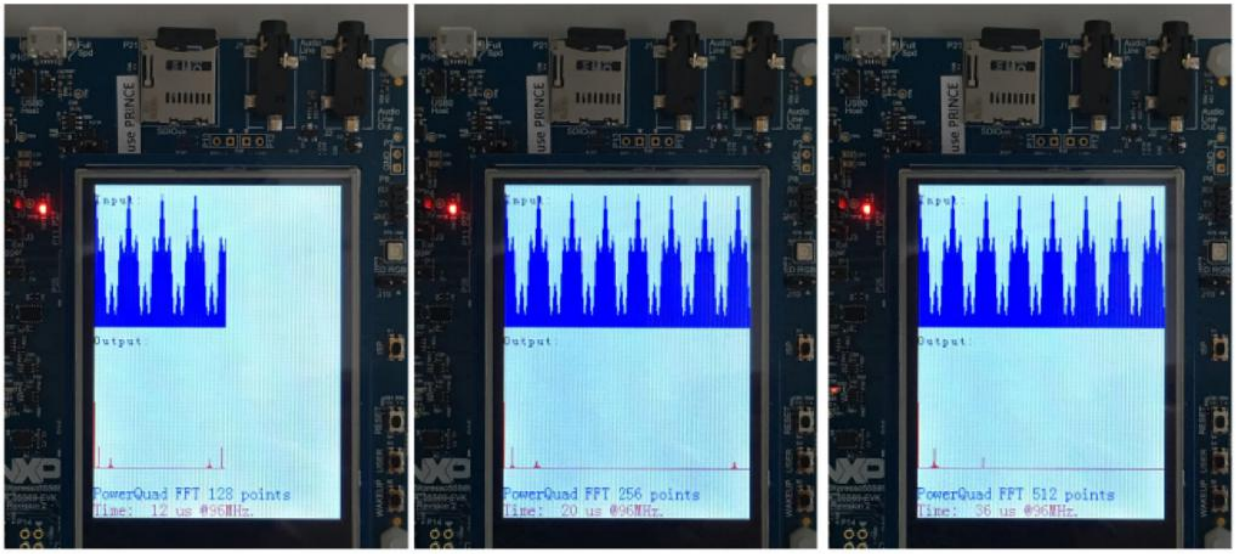

Figure 2. PowerQuad FFT 128/256/512 points

3.5 Matrix demo cases

The Matrix accelerator engine supports eight operations. Table 5 lists the operations and describes maximum supported dimensionality.

Table 5. PowerQuad matrix length range

PowerQuad engine	Operation	Max. row
Matrix	Addition	16 × 16
	Subtraction	16 × 16
	Hadamard product	16 × 16
	Product	16 × 16
	Vector dot-product	256 elements
	Inversion	9 × 9
	Transpose	16 × 16
	Scaling	16 × 16

Matrix data are stored in memory row-by-row, arranged like standard C/C++ arrays. So, if two 2 × 2 integer matrices A and B are:

```
A =    [1     2]    B =    [5     6]
           [3     4]           [7     8]
```

Then the input data is stored in memory arrays as follows:

```
int    MatA[4]    =    {1,    2,    3,    4};
int    MatB[4]    =    {5,    6,    7,    8};
```

Table 6 lists the usage of memory handlers for PowerQuad Matrix engine.

Table 6. Usage of memory handlers for Matrix engine

Operation	Driver function	Access type	Input/ Output data formats	Input A region usage	Input B region usage	Output region usage	Temp. region usage	Engine
Matrix addition	`Pq_mtx_add`	AHB	FP, Fix-16, Fix-32	Matrix M1	Matrix M2	Result matrix	N.A.	Matrix
Matrix substraction	`Pq_mtx_sub`	AHB	FP, Fix-16, Fix-32	Matrix M1	Matrix M2	Result matrix	N.A.	Matrix
Matrix hadamard product	`Pq_mtx_hadamard`	AHB	FP, Fix-16, Fix-32	Matrix M1	Matrix M2	Result matrix	N.A.	Matrix
Matrix product	`Pq_mtx_prod`	AHB	FP, Fix-16, Fix-32	Matrix M1	Matrix M2	Result matrix	N.A.	Matrix
Matrix invert	`Pq_mtx_inv`	AHB	FP, Fix-16, Fix-32	Matrix M1	N.A.	Result matrix	Max. 1024 words	Matrix
Matrix transpose	`Pq_mtx_tran`	AHB	FP, Fix-16, Fix-32	Matrix M1	N.A.	Result matrix	N.A.	Matrix
Matrix scale	`Pq_mtx_scale`	AHB	FP, Fix-16, Fix-32	Matrix M1	N.A. (scale factor in MISC register)	Result matrix	N.A.	Matrix
Vector dot product	`Pq_vec_dotp`	AHB	FP, Fix-16, Fix-32	Vector A	Vector B	Scaler result	N.A.	Matrix

In the demo case, there are three calculations used for each task:

- `task_pq_mat_add()` for matrix addition
- `task_pq_mat_mul()` for matrix multiplication
- `task_pq_mat_inv()` for matrix inversion

Just like the FFT, the PowerQuad driver implements the CMSIS-DSP API as well. Taking the `task_pq_mat_add()` as an example, the usage is the same as CMSIS-DSP API.

```
#define PQ_MAT_ROW_COUNT_MAX    16u
  #define PQ_MAT_COL_COUNT_MAX    16u
  /* A + B = C. */
  void task_pq_mat_add(void)
  {
  arm_matrix_instance_f32 matrixA;
  arm_matrix_instance_f32 matrixB;
  arm_matrix_instance_f32 matrixC;
```

```
  float32_t mDataA[PQ_MAT_ROW_COUNT_MAX][PQ_MAT_COL_COUNT_MAX];
  float32_t mDataB[PQ_MAT_ROW_COUNT_MAX][PQ_MAT_COL_COUNT_MAX];
  float32_t mDataC[PQ_MAT_ROW_COUNT_MAX][PQ_MAT_COL_COUNT_MAX];
  uint32_t i, j;
  uint32_t calcTime;

   /* Initialize the matrix. */
   for (i = 0u; i < PQ_MAT_ROW_COUNT_MAX; i++)
   {
   for (j = 0u; j < PQ_MAT_COL_COUNT_MAX; j++)
   {
    mDataA[i][j] = 1.0f * i * PQ_MAT_ROW_COUNT_MAX + j;
    mDataB[i][j] = 1.0f * i * PQ_MAT_ROW_COUNT_MAX + j;
    }
    }
    matrixA.numRows = PQ_MAT_ROW_COUNT_MAX; matrixA.numCols = PQ_MAT_COL_COUNT_MAX; matrixA.pData =
(float32_t *)mDataA; matrixB.numRows = PQ_MAT_ROW_COUNT_MAX; matrixB.numCols = PQ_MAT_COL_COUNT_MAX;
matrixB.pData = (float32_t *)mDataB; matrixC.numRows = PQ_MAT_ROW_COUNT_MAX; matrixC.numCols
= PQ_MAT_COL_COUNT_MAX;
    matrixC.pData    = (float32_t *)mDataC;
    /* Calc & Measure. */
    TimerCount_Start();
    arm_mat_add_f32(&matrixA, &matrixB, &matrixC);
    TimerCount_Stop(calcTime);

    /* Todo ...
    * - Record the time.
    * - Display the waveform.
    */
    }
```

When running the demo project, the display pages on the LCD screen module for each Matrix demo case are as shown in Figure 3.

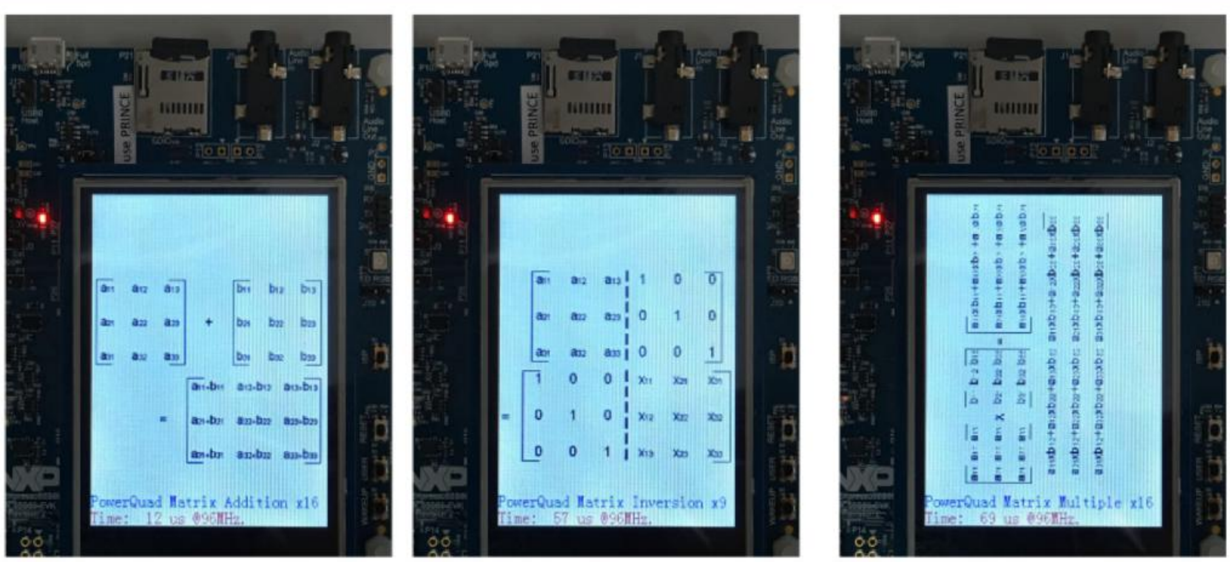

Figure 3. PowerQuad matrix addition/inversion/multiplication

3.6 FIR demo cases

The goal of this demonstration is to create a high-pass/low-pass FIR filter.

There are two demo cases to create different filters:

- `task_pq_fir_lowpass()` for low-pass filter, to remove the high frequency and get the low frequency from the mixed signal.
- `task_pq_fir_highpass()` for high-pass filter, to remove the low frequency and get the high frequency from the mixed signal.

In the demo cases, Matlab software calculates the taps (coefficients) for filters. Then into the PowerQuad, the hardware helps to do the filter process to signal automatically. Time consuming mathematical calculation is avoided.

The original signal is mixed with a low frequency signal (a sine wave at 1 kHz) and a high frequency signal (a sin wave at 15 kHz). Figure 4 is for waveform and Figure 5 is for frequency spectrum.

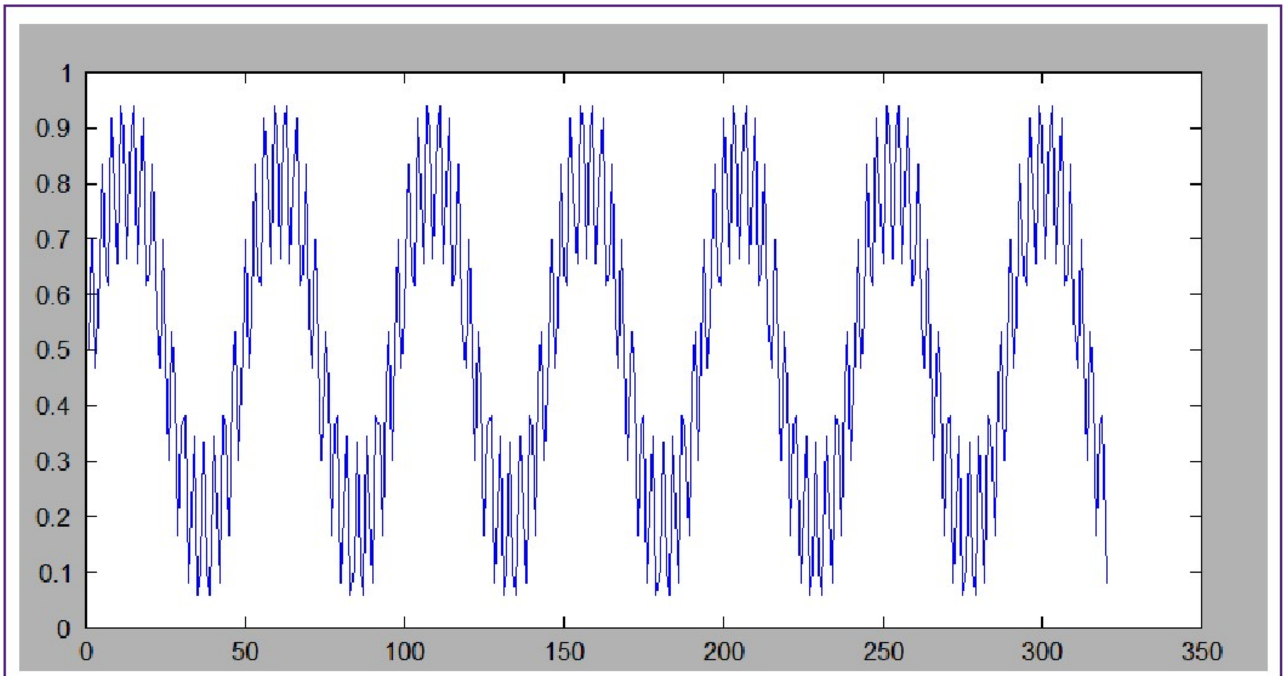

Figure 4. Waveform of mixed signal

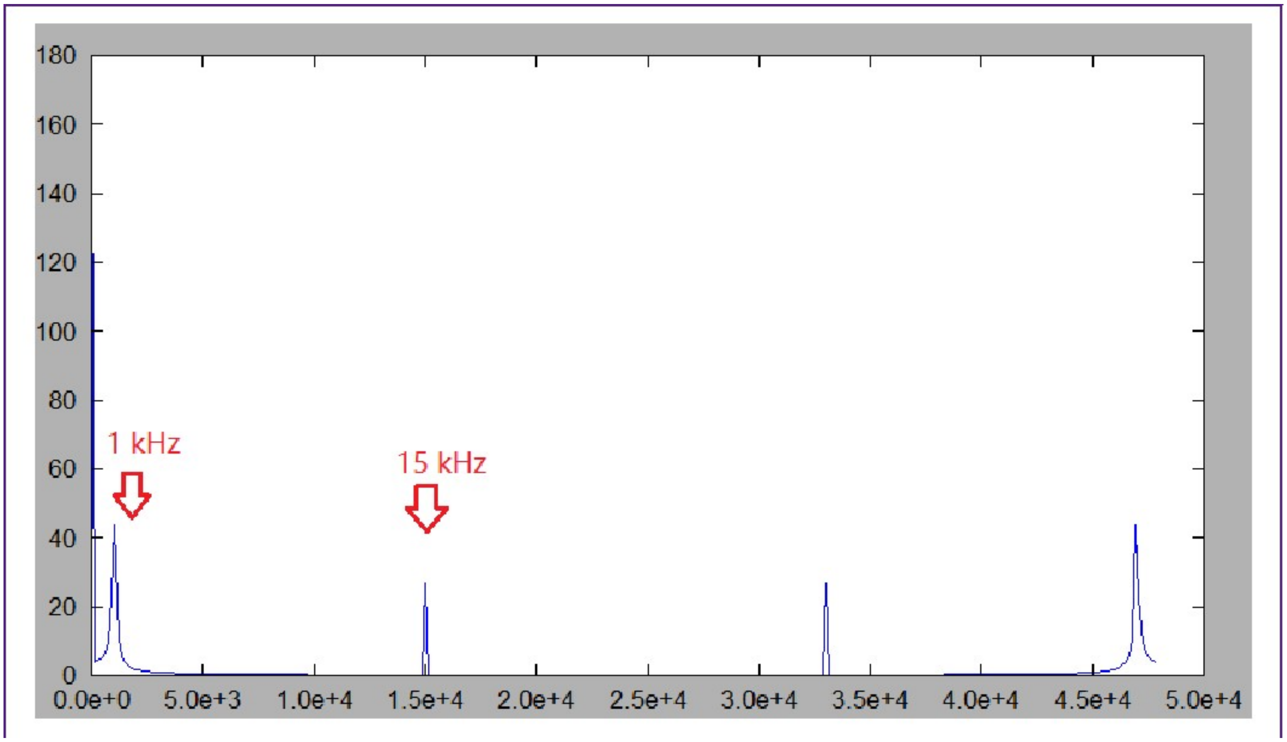

Figure 5. Frequency spectrum of mixed signal

To create the coefficients, run the following codes in MatLab.

```
clear all
close all
Fs=48000;
T=1/Fs;
Lenght=320;
t=(0:Lenght-1)*T;
Input_signal=(sin(2*pi*1000*t)+0.5*sin(2*pi*15000*t)+1.5)/3;
figure;
plot(Input_signal);
res=fft(Input_signal,Lenght);
figure;
f=((0:Lenght-1)/320*Fs);
plot(f,abs(res));
Cutoff_Freq=6000;
Nyq_Freq=Fs/2;
cutoff_norm=Cutoff_Freq/Nyq_Freq;
order=31;
FIR_Coeff=fir1(order,cutoff_norm,'high'); % for high-pass
%FIR_Coeff=fir1(order,cutoff_norm); % for low-pass
Filterd_signal=filter(FIR_Coeff,1,Input_signal);
figure;
plot(Filterd_signal);

fvtool(FIR_Coeff,'Fs',Fs); % generate the coeff and display the diagram
```

The filter features are:

- Type: high-pass/low-pass
- Order: 32

- Sampling frequency: 48 kHz
- Cut-off frequency: 6 kHz

Figure 6, Figure 7, Figure 8, and Figure 9 show the response reports.

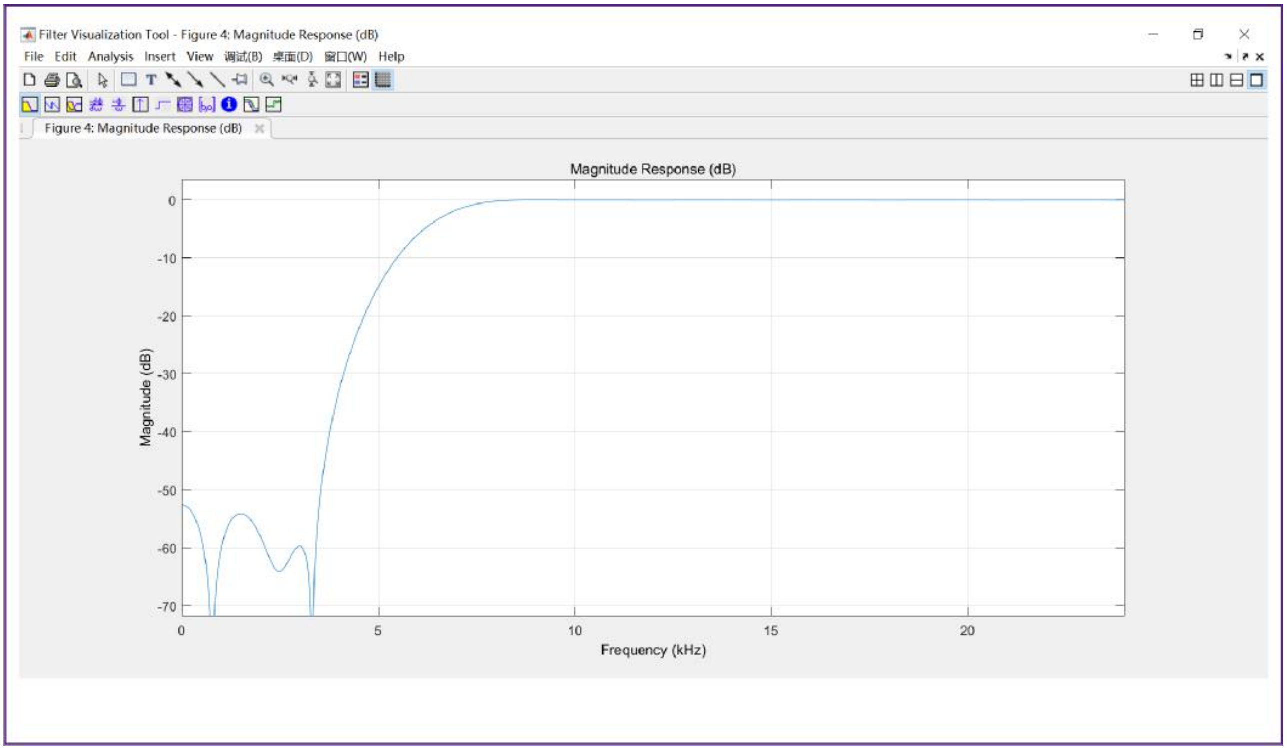

Figure 6. Magnitude response of FIR filter

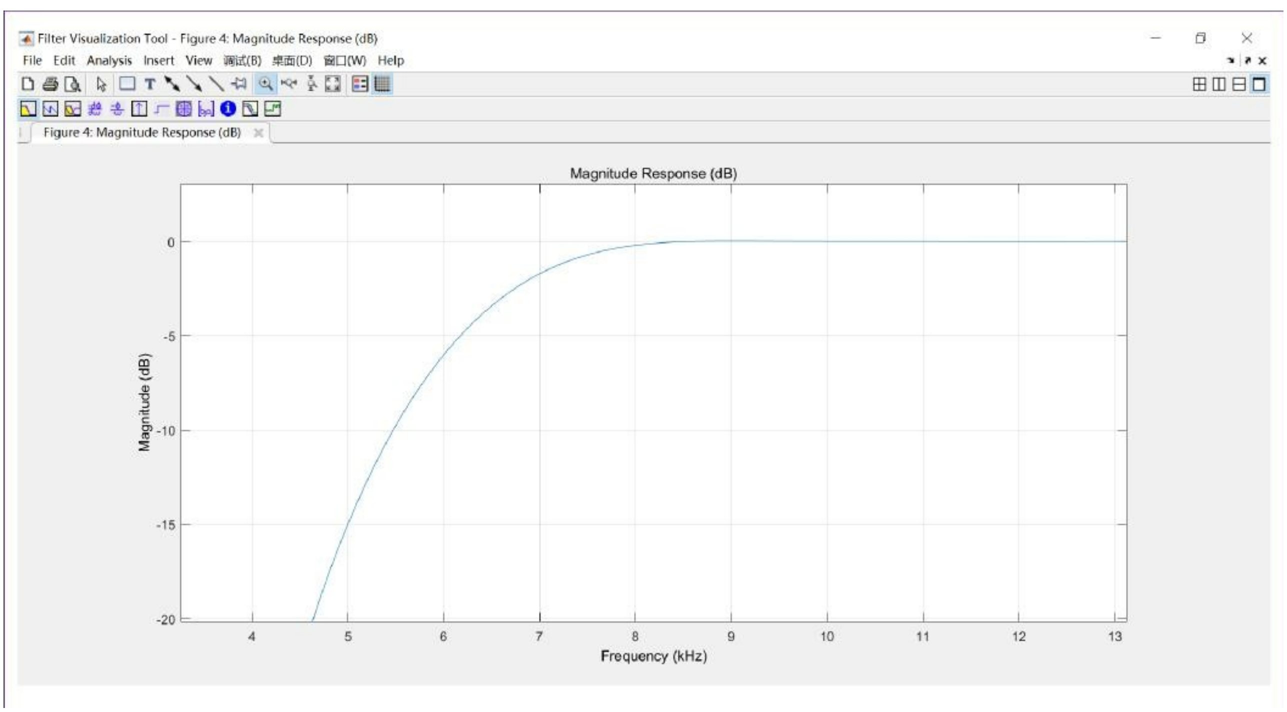

Figure 7. Magnitude response of FIR filter

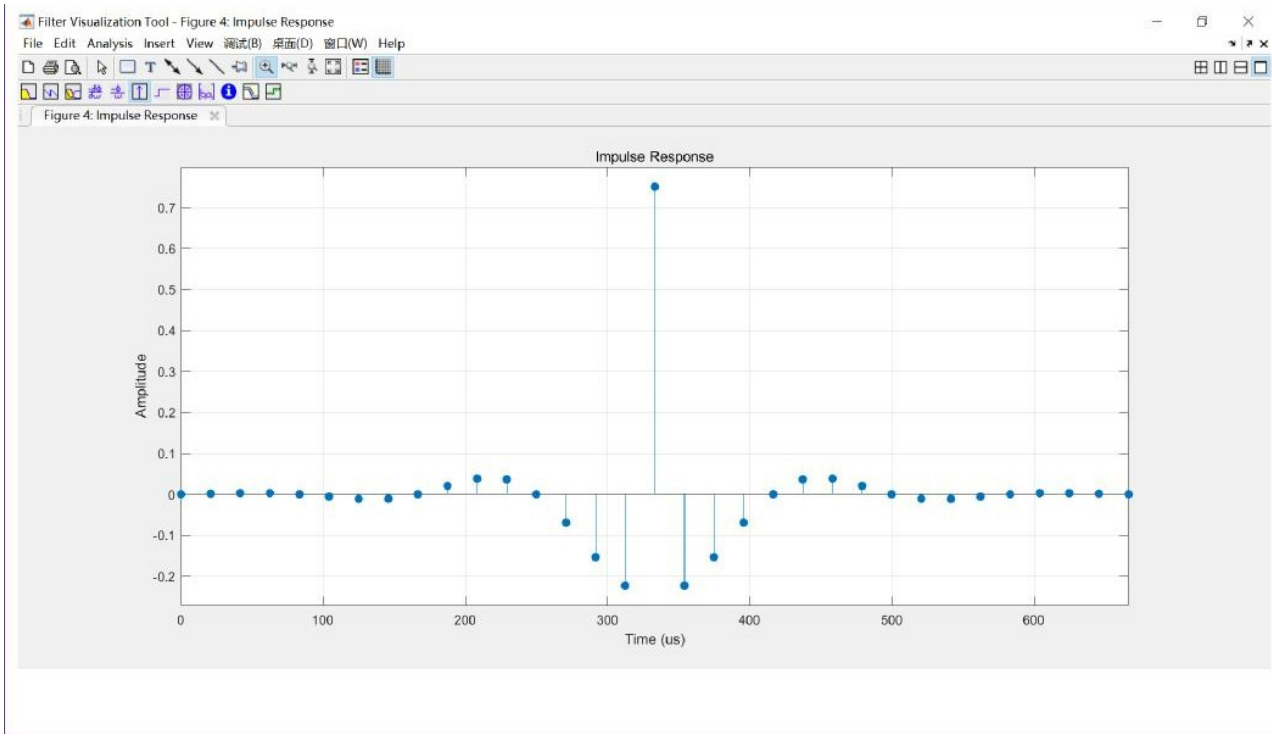

Figure 8. Impulse response of FIR filter

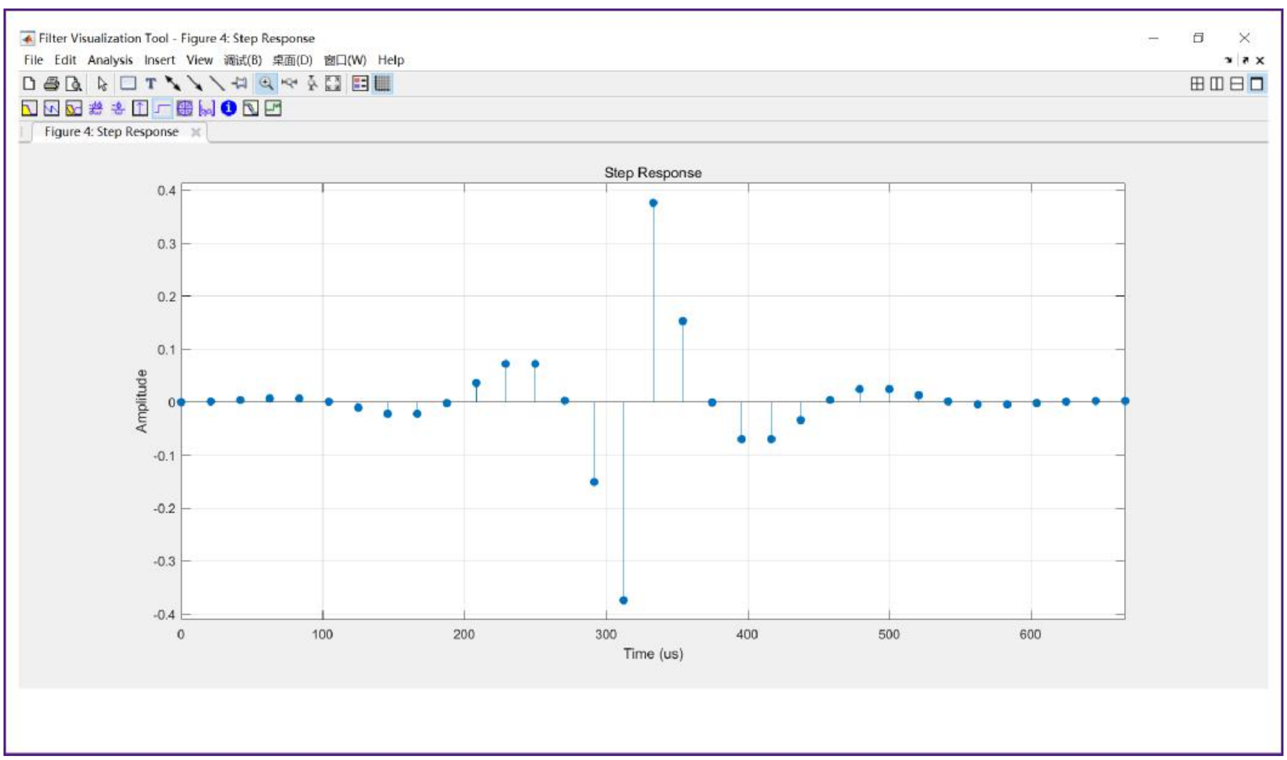

Figure 9. Step response of FIR filter

Set the PowerQuad to execute the filter process on MCU, taking a high-pass task as an example.

```
void task_pq_fir_highpass(void)
{
     uint32_t i;
     uint32_t Fs=48000;

     arm_fir_instance_f32 S;
     float32_t *inputF32, *outputF32;
     uint32_t calcTime;

     inputF32 = &gPQFirF32In[0];
     outputF32 = &gPQFirF32Out[0];

     /* Generate the wave. */
     for (i = 0; i < FIR_INPUT_LEN; i++)
     {
         gPQFirF32In[i] = 1.5
                         + 0.5 * arm_sin_f32(2*PI*15000*i/Fs)
                         +    arm_sin_f32(2*PI*1000*i/Fs) ;
         gPQFirF32In[i] /= 3.0f;
      }
      // ...

      /* Call FIR init function to initialize the instance structure. */
      arm_fir_init_f32(    &S,
                                NUM_TAPS,
                                (float32_t *)&firCoeffs32_highpass[0],
                                &firStateF32[0],
                                FIR_INPUT_LEN );
      PQ_Init(POWERQUAD_NS);
      pq_config_t pqConfig;

      pqConfig.inputAFormat = kPQ_Float;
      pqConfig.inputAPrescale = 0;
      pqConfig.inputBFormat = kPQ_Float;
      pqConfig.inputBPrescale = 0;
      pqConfig.outputFormat = kPQ_Float;
      pqConfig.outputPrescale = 0;
      pqConfig.tmpFormat = kPQ_Float;
      pqConfig.tmpPrescale = 0;
      pqConfig.machineFormat = kPQ_Float;
      pqConfig.tmpBase = (uint32_t *)0xE0000000;
      PQ_SetConfig(POWERQUAD_NS, &pqConfig);

      /* move the taps into private RAM to improve the performance of operating memory. */
      PQ_MatrixScale( POWERQUAD_NS,
                      POWERQUAD_MAKE_MATRIX_LEN(16, NUM_TAPS / 16, 0),
                      1.0,
                      firCoeffs32_highpass,
                      EXAMPLE_PRIVATE_RAM );
      PQ_WaitDone(POWERQUAD_NS);

      /* In the next calculation, data in private ram is used. */
      pqConfig.inputBFormat = kPQ_Float;
      pqConfig.outputFormat = kPQ_Float;
      PQ_SetConfig(POWERQUAD_NS, &pqConfig);

      TimerCount_Start();
      PQ_FIR(POWERQUAD_NS, inputF32, APP_PQ_FIR_SAMPLE_COUNT_240, EXAMPLE_PRIVATE_RAM, NUM_TAPS,
```

```
outputF32,PQ_FIR_FIR);
      PQ_WaitDone(POWERQUAD_NS);
      //arm_fir_f32(&S, inputF32, outputF32, FIR_INPUT_LEN);
      TimerCount_Stop(calcTime);

      /* Todo ...
      * - Record the time.
      * - Display the waveform.
      */
}
```

When running the demo cases to execute the filter with PowerQuad hardware, the results are shown in the LCD Screen, as shown in Figure 10.

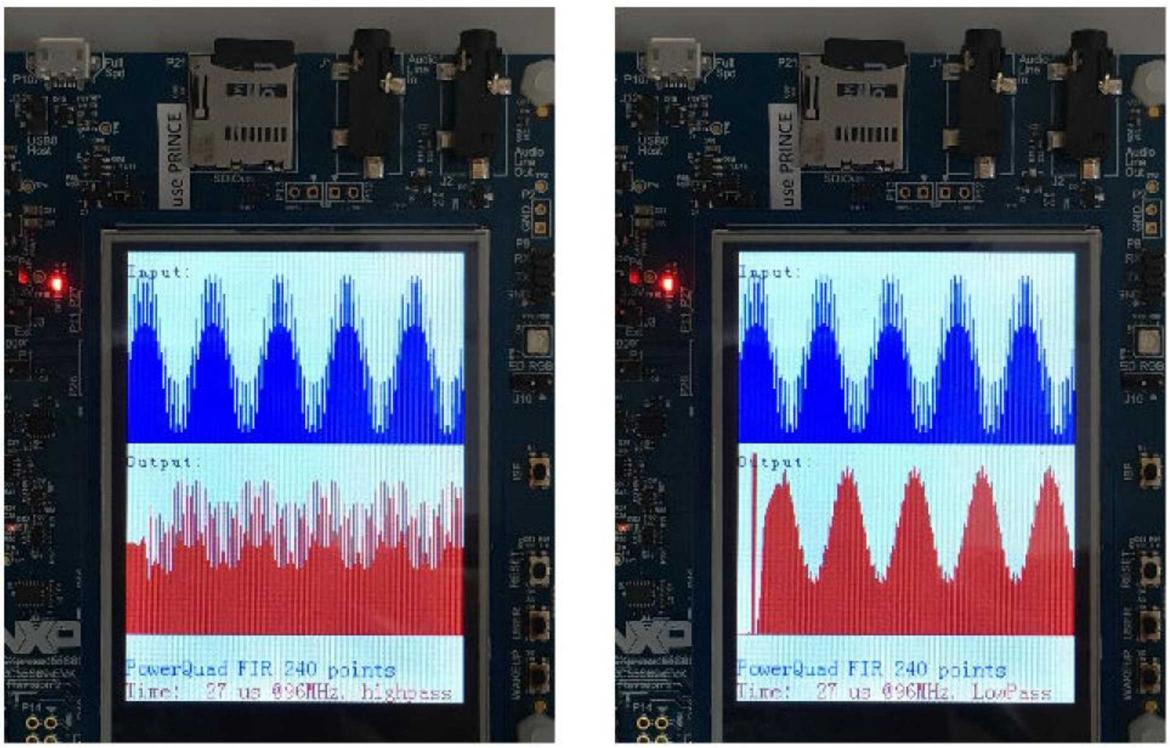

Figure 10. PowerQuad FIR High-Pass/Low-Pass filter

4 PowerQuad vs Arm CMSIS-DSP performance

In the demo project, a page is set up for the comparison between the PowerQuad and Arm CMSIS-DSP when they are running the same tasks. To make a fair comparison, when running the DSP task, to achieve the highest performance, run the Arm CMSIS-DSP code in RAM and use the dedicated RAM (the private one) for PowerQuad.

Figure 11 shows the snapshot of the screen.

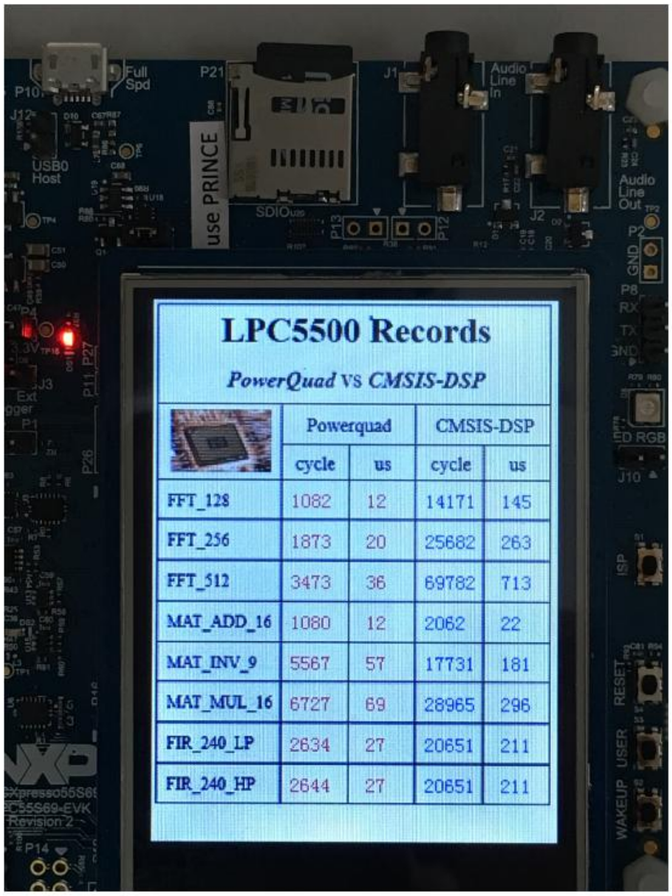

Figure 11. Running time for PowerQuad vs Arm CMSIS-DSP

Figure 12 summarizes the data.

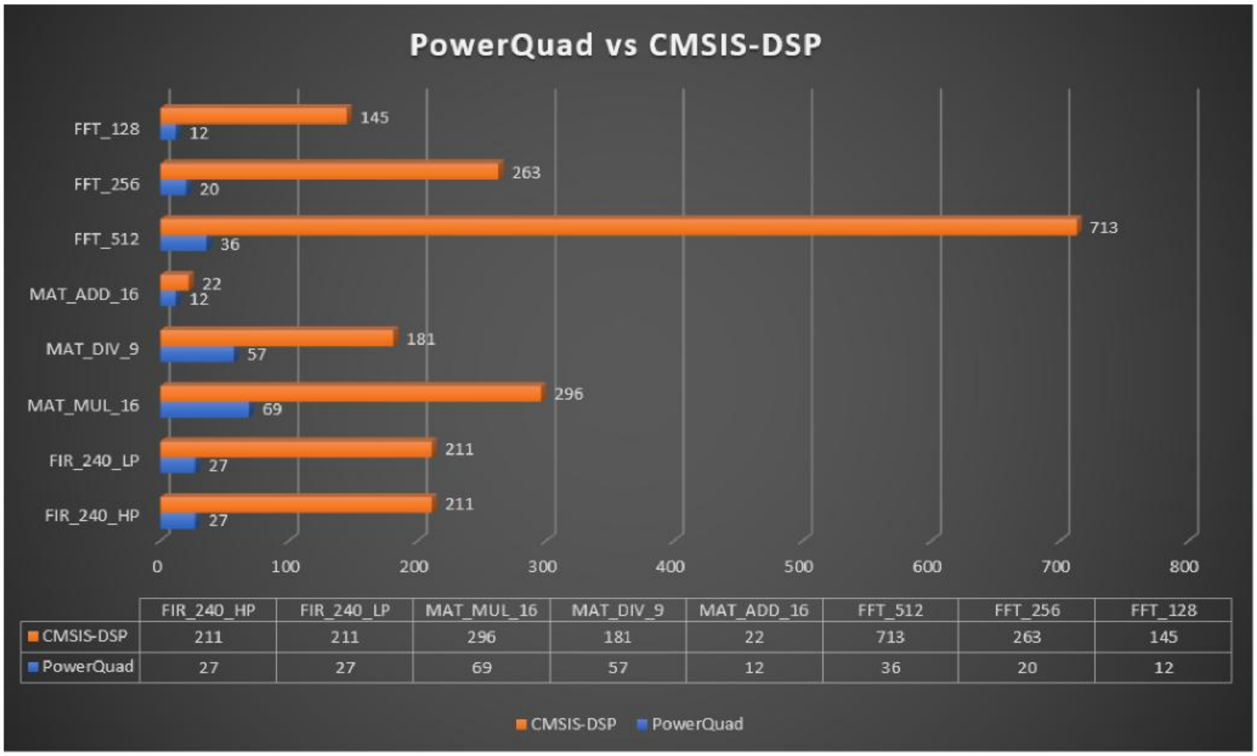

Figure 12. Running time data table for PowerQuad vs Arm CMSIS-DSP

5 Revision history

Rev.	Date	Description
0	25 December 2021	Initial release
1	25 May 2022	Replace LPC55S36 with LPC553x/LPC55S3x

Legal information

Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at http://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this data sheet expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, μVision, Versatile — are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved.

Airfast — is a trademark of NXP B.V.

Bluetooth — the Bluetooth wordmark and logos are registered trademarks owned by Bluetooth SIG, Inc. and any use of such marks by NXP Semiconductors is under license.

Cadence — the Cadence logo, and the other Cadence marks found at www.cadence.com/go/trademarks are trademarks or registered trademarks of Cadence Design Systems, Inc. All rights reserved worldwide.

CodeWarrior — is a trademark of NXP B.V.

ColdFire — is a trademark of NXP B.V.

ColdFire+ — is a trademark of NXP B.V.

EdgeLock — is a trademark of NXP B.V.

EdgeScale — is a trademark of NXP B.V.

EdgeVerse — is a trademark of NXP B.V.

eIQ — is a trademark of NXP B.V.

FeliCa — is a trademark of Sony Corporation.

Freescale — is a trademark of NXP B.V.

HITAG — is a trademark of NXP B.V.

ICODE and I-CODE — are trademarks of NXP B.V.

Immersiv3D — is a trademark of NXP B.V.

I2C-bus — logo is a trademark of NXP B.V.

Kinetis — is a trademark of NXP B.V.

Layerscape — is a trademark of NXP B.V.

Mantis — is a trademark of NXP B.V.

MIFARE — is a trademark of NXP B.V.

MOBILEGT — is a trademark of NXP B.V.

NTAG — is a trademark of NXP B.V.

Processor Expert — is a trademark of NXP B.V.

QorIQ — is a trademark of NXP B.V.

SafeAssure — is a trademark of NXP B.V.

SafeAssure — logo is a trademark of NXP B.V.

StarCore — is a trademark of NXP B.V.

Synopsys — Portions Copyright © 2021 Synopsys, Inc. Used with permission. All rights reserved.

Tower — is a trademark of NXP B.V.

UCODE — is a trademark of NXP B.V.

VortiQa — is a trademark of NXP B.V.

Date of release: 25 May 2022

Document identifier: AN13498