CMP103

Edge Intelligence: How to Leverage Silicon Labs AI/ML to Improve Efficiency and Performance

Rich Lysaght

Senior Product Marketing Manager

SILICON LABS

TinyML vs AI/ML

AI/ML - Broader machine learning models that require more computational power and are typically run on servers or high-performance edge devices.

- Complex tasks such as deep learning, large-scale data analysis, and sophisticated pattern recognition.
- Greater accuracy, advanced capabilities, and ability to handle complex data processing tasks.

TinyML - A subset of machine learning designed for deployment on microcontrollers and low-power devices.

- Ideal for simple tasks like anomaly detection and basic classification in resource-constrained environments.
- Low power consumption, real-time processing on-device, and cost-effective for embedded applications.

Why Machine Learning on Microcontrollers?

Reduce Decision Latency

- Make more real time decisions closer to where the data is collected

Lower data and device security risk

- Keeping data local to devices reduces risk of exposure during transmission

Bandwidth Constraints

- Bandwidth limited IoT networks cannot transmit large amounts of data required for cloud centric architectures

Offline Mode Operation

- Allows for nodes to operate autonomously and make decisions even when network is unavailable

Lower Device and Service Cost

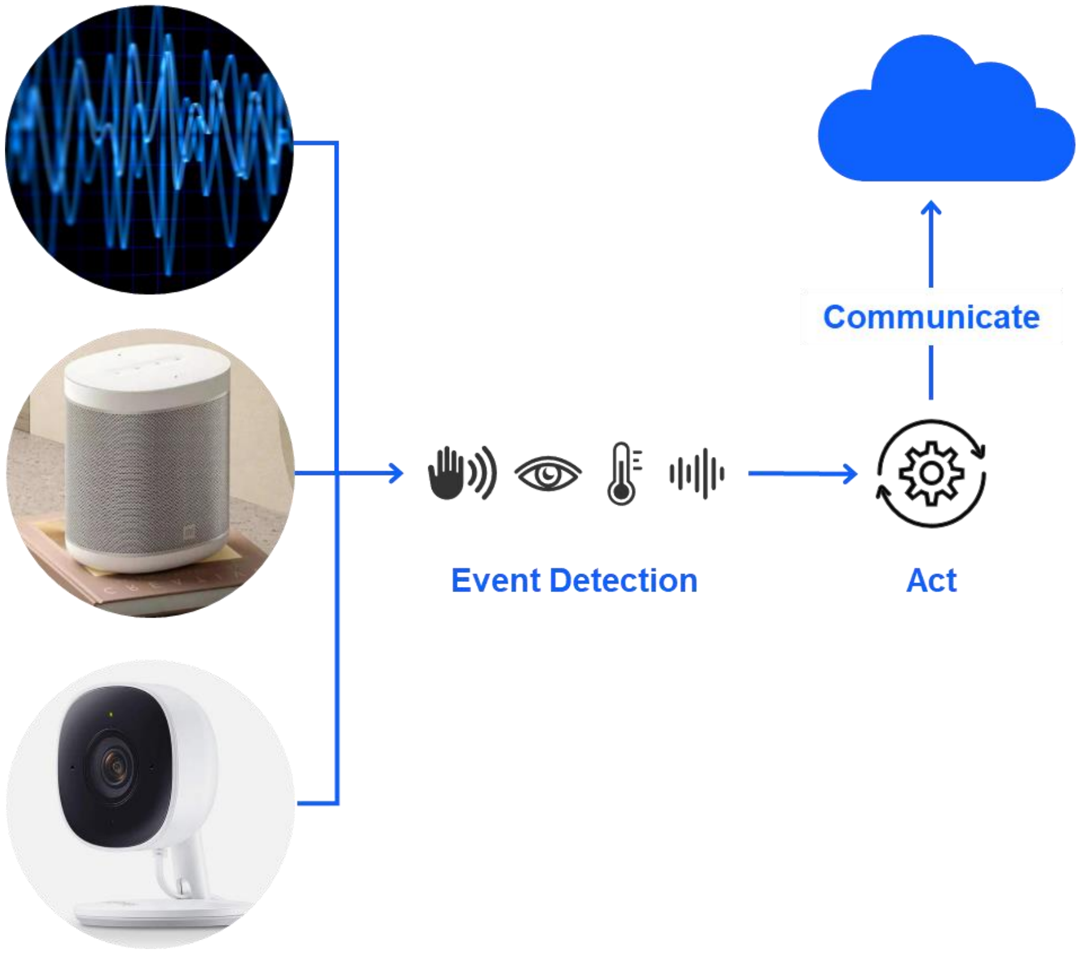
- Lowers performance requirements for sensor devices and limits recurring costs

Improved Low Power Operation

- Reduces number of network transmissions to improve overall battery life

Data processing is more efficient with Machine Learning at the sensor level

Reducing Decision Latency



More Real-Time Responses:

- Rapid decision-making allows systems to respond instantly to changes.

Increased Accuracy:

- Quick decisions based on real-time data help in making timely adjustments, leading to more accurate outcomes

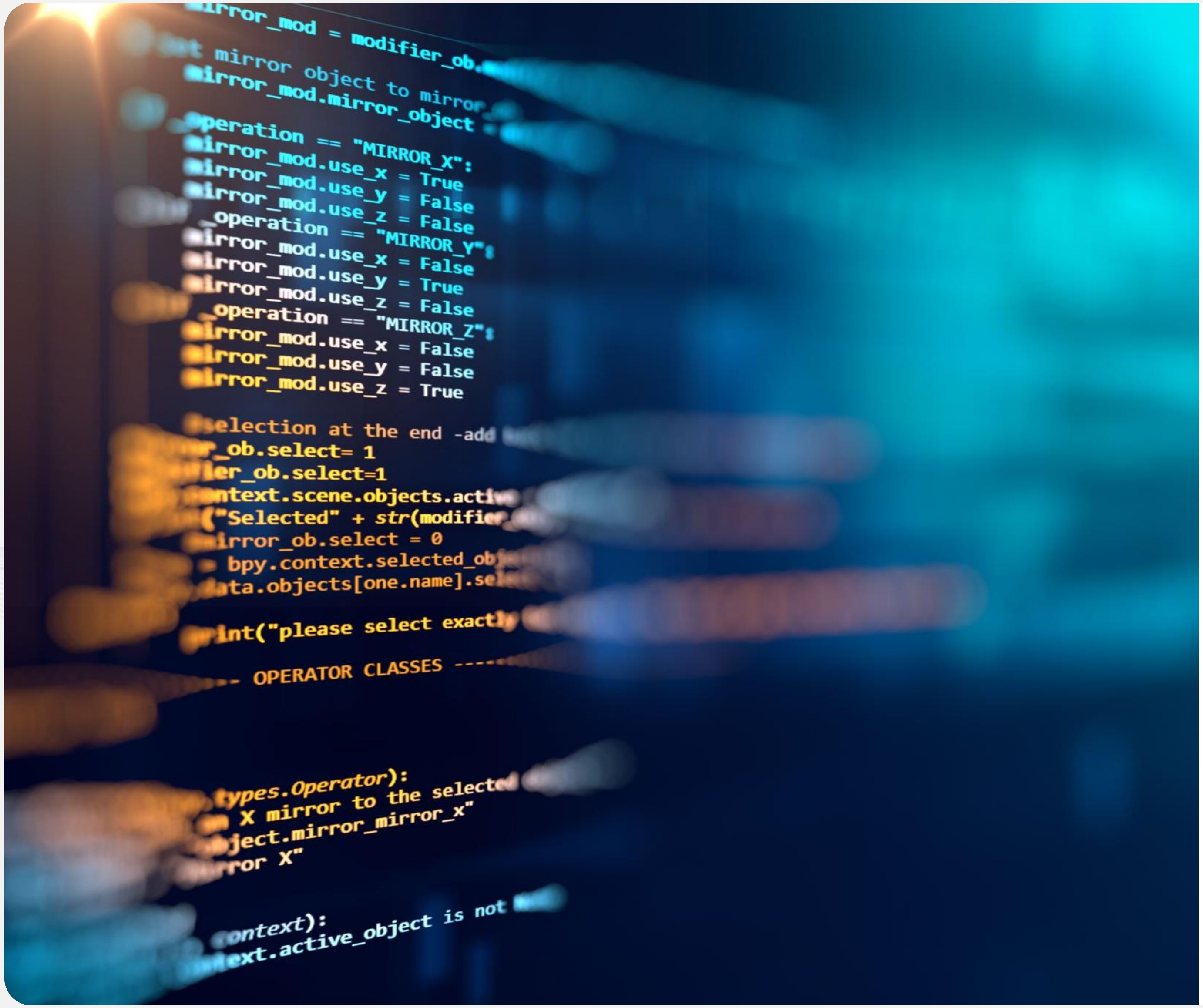
Adaptive Learning locally:

- Systems can continuously learn and adapt to new data more effectively, improving model accuracy and reliability over time.

Parallel Processing Capabilities:

- Addition of AIML accelerators enable simultaneous task execution, resulting in higher throughput and faster response times for real-time applications

Lowering Data Security Risks



Local Data Processing:

- Reduces the need to transmit sensitive information over networks, minimizing exposure to interception.
- Results are sent to the cloud rather than the data.

Privacy Compliance:

- Keeping data on-device aligns with privacy regulations, as less personal data is transmitted or stored in the cloud, thereby minimizing compliance risks.

Data Encryption:

- Secure Vault with PSA L3 certification ensures sensitive data to be encrypted before storage or transmission, adding an additional layer of security.

Firmware Updates:

- Secure over-the-air (OTA) updates for firmware can help ensure that devices are running the latest security protocols and patches, protecting against vulnerabilities.

Real-Time Anomaly Detection:

- TinyML can continuously monitor for unusual patterns locally, enabling immediate responses to potential threats.

Addressing Bandwidth Limitations of IoT Networks

Reduce Data Volume:

- Only relevant data is transmitted instead of raw data. This is crucial in low-bandwidth settings like LPWAN, which typically supports data rates of 0.1 to 50 kbps.

Optimize Use of Limited Bandwidth:

- Given LPWAN’s constraints, minimizing the data sent not only conserves bandwidth but also enhances the reliability of communication.

Selective Data Transmission:

- Edge devices send updates only when specific conditions are met (e.g., threshold breaches), which is crucial for LPWAN, where frequent transmissions can quickly exhaust bandwidth.

Scalability:

- BLE mesh allows devices to communicate directly, reducing reliance on a central hub and enabling a scalable network where many devices can relay information.
- Wi-Fi networks effectively segment traffic, and by processing data at the edge, devices reduce network load and improve efficiency, allowing for better scaling without performance degradation.

Making Decisions Without Network Access

Self-Sufficient Devices:

- Edge devices can operate independently, making them ideal for applications in remote or challenging locations where connectivity may be limited.

Reduced Infrastructure Cost:

- Fewer data transmissions lead to lower costs for cloud services and bandwidth, making IoT solutions more economical.

Removes Dependence on Cloud Processing:

- Without offline capabilities, devices rely on stable internet connectivity, which can lead to downtime and reduced effectiveness in areas with poor connectivity.

Lower Risk of Data Loss:

- Unstable connections can result in data being lost or corrupted during transmission, compromising the integrity of the information and potentially leading to poor decisions.

Lowering Costs to Make AI/ML More Accessible

Affordable Hardware:

- Advancements in technology have created cheaper, more efficient hardware like MCUs with accelerators for running AI/ML algorithms, reducing initial investments and broadening application possibilities.

Wider Adoption:

- More entities can integrate AI/ML into their operations, enhancing innovation and competition across sectors.

Diverse Applications:

- Smaller companies and startups can leverage AI/ML for various applications, from automation to data analysis, driving economic growth.

Reduced Operating Expenses:

- Streamlined algorithms and optimized hardware lead to lower energy consumption and maintenance costs, decreasing ongoing operational expenses.

Increased ROI:

- Lower recurring costs improve the return on investment for AI/ML projects, making them more attractive for businesses.

Sustainability:

- Reduced energy and operational costs contribute to more sustainable practices, appealing to environmentally conscious organizations.

Optimized Performance for Low Power Devices

Compact Design:

- Lower power requirements enable smaller batteries or energy storage solutions, leading to more compact device designs, which is crucial in space-limited applications like wearables and small sensors.

Efficient Algorithms:

- Edge AI/ML models are optimized for resource-constrained environments through techniques like quantization and pruning, reducing computational complexity and lowering energy consumption.

Adaptive Sampling:

- Edge devices use adaptive sampling to collect and process data only when needed, minimizing unnecessary computations and data transfers, thereby conserving power.

Extended Battery Life:

- Reduced power consumption results in longer battery life, decreasing the need for replacements or recharging, which lowers maintenance efforts and costs, enhancing user-friendliness.

Network Efficiency:

- Low-power devices can operate effectively in diverse environments and network conditions, enabling scalable IoT solutions without overwhelming infrastructure.
- Enhanced Scalability:
- Efficient Resource Utilization, local processing reduces the need for centralized resources, allowing for more IoT devices to be deployed without overwhelming network infrastructure.

Adding Local Acceleration for AI/ML Inferencing

MVP Math library

Accelerate and do more efficient linear algebra operations with internal MVP subsystem

Math **APIs (alternative to CMSIS_DSP)** available in GSDK

VECTOR OPERATIONS

- Vector Add
- Vector Absolute Value
- Vector Clip
- Vector Dot Product
- Vector Multiply
- Vector Negate
- Vector Offset
- Vector Scale
- Vector Sub
- Complex Vector Conjugate
- Complex Vector Dot Product
- Complex Vector Magnitude
- Complex Vector Magnitude Squared
- Complex Vector Multiply
- Complex Vector Multiply Real
- Vector Copy
- Vector Fill

MATRIX OPERATIONS

- Matrix Initialize
- Matrix Multiply
- Matrix Scale
- Matrix Sub
- Matrix Transpose
- Matrix Multiply Vector
- Matrix Add
- Complex Matrix Multiply
- Complex Matrix Transpose

✓ **Faster and more efficient execution of many algorithms with large data for example filtering algorithms**
✓ **Saving CPU cycles, saving power, resulting longer battery life**
✓ **Option to win sockets against faster CPUs**

CortexM only

Matrix dims.		CMSIS f32 cpu-cycles	CMSIS f16 cpu-cycles	MVP cpu-cycles	instr	stalls
2x2	2x2	226	304	403	8	0
4x2	2x4	602	913	424	32	0
6x2	2x6	1210	1921	464	72	0
8x2	2x8	2050	3321	516	128	0
10x2	2x10	3122	5113	592	200	0
12x2	2x12	4426	7297	676	288	0
14x2	2x14	5962	9873	784	392	0
16x2	2x16	7730	12841	904	512	0
18x2	2x18	9730	16201	1036	648	0
20x2	2x20	11962	19953	1192	800	0
20x4	4x20	17962	27956	1593	1200	1
20x6	6x20	23742	39956	2193	1600	201
20x8	8x20	27562	47556	2793	2000	400
20x10	10x20	33162	59556	3393	2400	601
20x12	12x20	37162	67156	3993	2800	801
20x14	14x20	42762	79156	4593	3200	1000
20x16	16x20	46762	86756	5193	3600	1201
20x18	18x20	52362	98756	5793	4000	1401
20x20	20x20	56362	106356	6393	4400	1600

~ 9x less cycles

Benefits of the MVP ML Hardware Accelerator

Dedicated ML computing subsystem next to the CPU: Matrix Vector Processor (MVP)

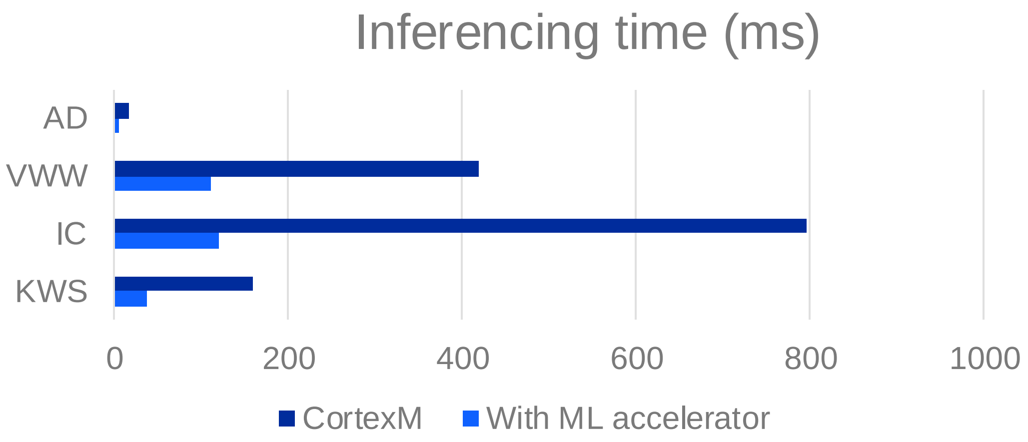
Optimized MVP to accelerate ML inferencing with a lot of processing power **offloading the CPU**

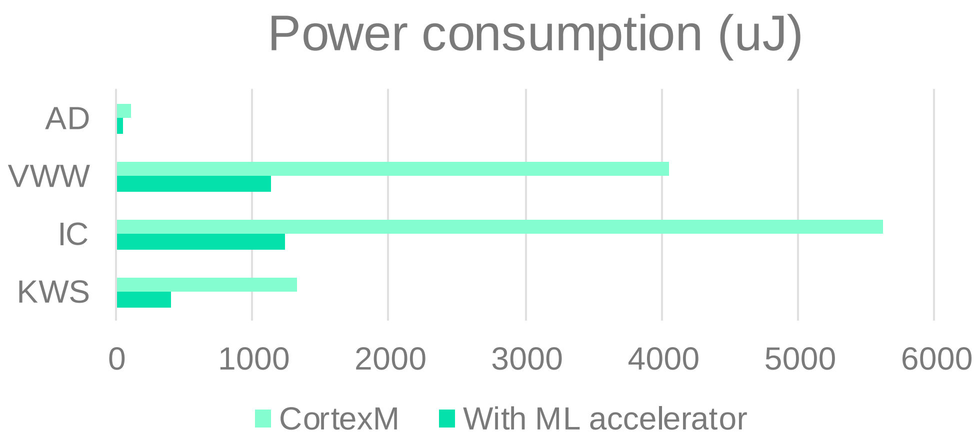
Up to 8x faster inferencing over Cortex-M (see below perf. benchmark)

Up to **6x lower power** for inferencing (see below perf. benchmark)

Dedicated OPNs for MVP accelerated parts → EFR32MG24B[2]… or [3]

Performance data with ML hardware accelerator vs. pure SW on CortexM*





*Standardized performance benchmark validated by independent benchmarking body **MLCommons.org**. Published in MLPerf Tiny v1.0. Results are for inferencing only (not for the complete application). You can refer to MLCommons as validated results-

ML Commons

MVP – Matrix Vector Processor Demo

MVP – Matrix Vector Processor (AI/ML Accelerator)

AI/ML Hardware Accelerator Key Features

Matrix Processor Accelerates ML Inferences

- Multi-dimensional array operations
- Handles real and complex data
- Offloads MCU

Up to 8x faster inference over Cortex-M

- Lower latency

Up to 6x lower power for inferencing

- Longer battery life

MVP Math Library

- Can be used for non-ML applications

- AI/ML Hardware Accelerator enables efficient Edge ML inferencing

MVP Math Library API

https://docs.silabs.com/d/platform-compute-math/4.3/

MVP MATH LIBRARY API

Vector Functions

- sl_math_mvp_vector_clip_f16
- sl_math_mvp_complex_vector_dot_product_f16
- sl_math_mvp_vector_copy_f16
- sl_math_mvp_vector_sub_f6
- sl_math_mvp_vector_mult_f16
- sl_math_mvp_vector_abs_f16
- sl_math_mvp_vector_scale_f16
- sl_math_mvp_vector_add_f16
- sl_math_mvp_vector_add_i8
- sl_math_mvp_complex_vector_mult_real_f16
- sl_math_mvp_complex_vector_mult_f16
- sl_math_mvp_vector_negate_f16
- sl_math_mvp_complex_vector_conjugate_f16
- sl_math_mvp_vector_fill_f16
- sl_math_mvp_complex_magnitude_squared_f16
- sl_math_mvp_vector_dot_product_f16
- sl_math_mvp_clamp_i8
- sl_math_mvp_vector_offset_f16

MVP MATH LIBRARY API

Matrix Functions

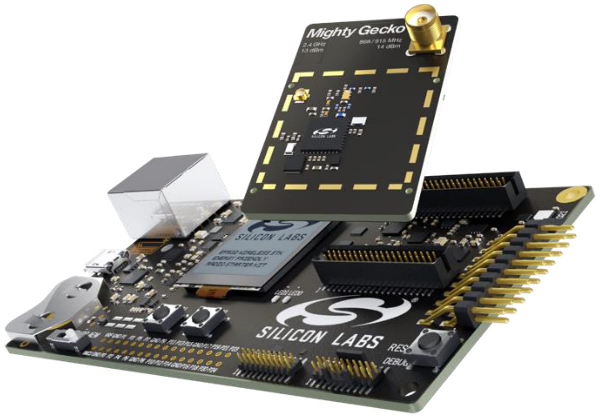
- sl_math_mvp_matrix_mult_f16
- sl_math_mvp_matrix_scale_f16
- sl_math_mvp_matrix_transpose_f16
- sl_math_mvp_complex_matrix_transpose_f16
- sl_math_mvp_matrix_add_f16
- sl_math_mvp_matrix_sub_f16
- sl_math_mvp_matrix_init_f16
- sl_math_mvp_matrix_vector_mult_f16
- sl_math_mvp_complex_matrix_mult_f16

Utility Functions

- sl_math_mvp_clear_errors
- sl_math_mvp_get_error

Demo

MVP Math Library

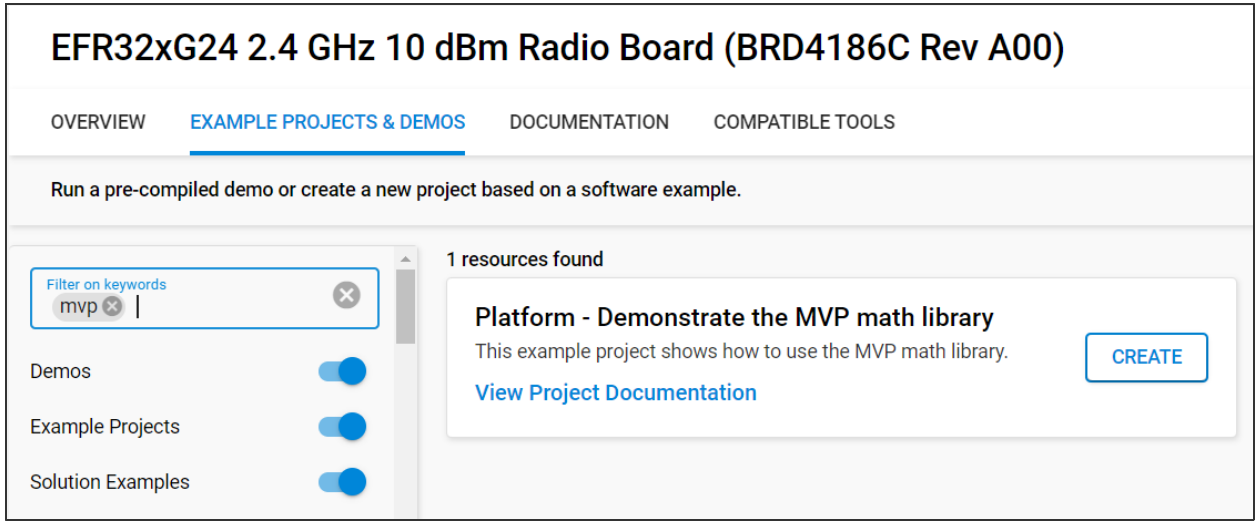


xG24-RB4186

- EFR32MG24B210F1536IM48
- +10dBm
- 1536kB Flash
- 256kB RAM
- MVP Equipped

WSTK

- SLWMB4002A



Demo

MVP Math Library Demo Multiply

```
printf("Fill matrix A with values:\n");
input_a[0] = 1.0;
input_a[1] = 2.0;
input_a[2] = 3.0;
input_a[3] = -4.0;
input_a[4] = 5.0;
input_a[5] = 6.0;
input_a[6] = 7.0;
input_a[7] = -8.0;
input_a[8] = 9.0;
input_a[9] = 10.0;
input_a[10] = 11.0;
input_a[11] = -12.0;
sl_math_matrix_init_f16(&matrix_a, 3, 4, input_a);
print_matrix(&matrix_a);

printf("Transpose matrix A into matrix B:\n");
sl_math_matrix_init_f16(&matrix_b, 4, 3, input_b);
sl_math_mvp_matrix_transpose_f16(&matrix_a, &matrix_b);
print_matrix(&matrix_b);

printf("Multiply matrix A with matrix B:\n");
sl_math_matrix_init_f16(&matrix_z, 3, 3, output);
sl_math_mvp_matrix_mult_f16(&matrix_a, &matrix_b, &matrix_z);
print_matrix(&matrix_z);
```

Demo

MVP Math Library Demo Output

COM4 - Tera Term VT

File Edit Setup Control Window Help

```
r functions
  will work for matrixes as well.

Fill vector A with 1.0:
  (  1.00),  (  1.00),  (  1.00),  (  1.00),  (  1.00),  (  1.00),
Fill vector B with -3.0:
  ( -3.00),  ( -3.00),  ( -3.00),  ( -3.00),  ( -3.00),  ( -3.00),
Add vector A and B:
  ( -2.00),  ( -2.00),  ( -2.00),  ( -2.00),  ( -2.00),  ( -2.00),

Fill vector A with complex values:
  [  1.00,  2.00],  [  3.00, -4.00],  [  5.00,  6.00],  [  7.00, -8.00],
Complex conjugate vector A into vector B:
  [  1.00, -2.00],  [  3.00,  4.00],  [  5.00, -6.00],  [  7.00,  8.00],
Complex multiplicate vector A with vector B:
  [  5.00,  0.00],  [ 25.00,  0.00],  [ 61.00,  0.00],  [113.00,  0.00],
Add vector A to the result:
  [  6.00,  2.00],  [ 28.00, -4.00],  [ 66.00,  6.00],  [120.00, -8.00],
Clip the result to -5..10:
  [  6.00,  2.00],  [ 10.00, -4.00],  [ 10.00,  6.00],  [ 10.00, -5.00],

Fill matrix A with values:
  (  1.00),  (  2.00),  (  3.00),  ( -4.00),
  (  5.00),  (  6.00),  (  7.00),  ( -8.00),
  (  9.00),  ( 10.00),  ( 11.00),  (-12.00),
Transpose matrix A into matrix B:
  (  1.00),  (  5.00),  (  9.00),
  (  2.00),  (  6.00),  ( 10.00),
  (  3.00),  (  7.00),  ( 11.00),
  ( -4.00),  ( -8.00),  (-12.00),
Multiply matrix A with matrix B:
  ( 30.00),  ( 70.00),  (110.00),
  ( 70.00),  (174.00),  (278.00),
  (110.00),  (278.00),  (446.00),

Fill matrix A with complex values:
  [  1.00,  1.50],  [  2.00, -2.50],  [  3.00,  3.50],  [  4.00, -4.50],
  [  5.00,  5.50],  [  6.00, -6.50],  [  7.00,  7.50],  [  8.00, -8.50],
  [  9.00,  9.50],  [ 10.00,-10.50],  [ 11.00, 11.50],  [ 12.00,-12.50],
Conjugate matrix A into matrix B:
  [  1.00, -1.50],  [  2.00,  2.50],  [  3.00, -3.50],  [  4.00,  4.50],
  [  5.00, -5.50],  [  6.00,  6.50],  [  7.00, -7.50],  [  8.00,  8.50],
  [  9.00, -9.50],  [ 10.00, 10.50],  [ 11.00,-11.50],  [ 12.00, 12.50],
Complex transpose matrix B into matrix C:
  [  1.00, -1.50],  [  5.00, -5.50],  [  9.00, -9.50],
  [  2.00,  2.50],  [  6.00,  6.50],  [ 10.00, 10.50],
  [  3.00, -3.50],  [  7.00, -7.50],  [ 11.00,-11.50],
  [  4.00,  4.50],  [  8.00,  8.50],  [ 12.00, 12.50],
Complex multiply matrix A with matrix C:
  [ 71.00,  0.00],  [159.00,  0.00],  [247.00,  0.00],
  [159.00,  0.00],  [375.00,  0.00],  [591.00,  0.00],
  [247.00,  0.00],  [591.00,  0.00],  [935.00,  0.00],

Demonstrate that none of the previous math library functions gave any errors or
exceptions:
  Execution status: expected=0, actual=0
```

Demo

Matrix Multiply Example

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 10 & 11 \\ 20 & 21 \\ 30 & 31 \end{bmatrix}$$

$$= \begin{bmatrix} 1x10 + 2x20 + 3x30 & 1x11 + 2x21 + 3x31 \\ 4x10 + 5x20 + 6x30 & 4x11 + 5x21 + 6x31 \end{bmatrix}$$

$$= \begin{bmatrix} 10+40+90 & 11+42+93 \\ 40+100+180 & 44+105+186 \end{bmatrix} = \begin{bmatrix} 140 & 146 \\ 320 & 335 \end{bmatrix}$$

Demo

Matrix Multiply Example - Initialization

```c
//Variables for RGL matrix math
static float16_t    rgl_input_a[4][4];
static float16_t    rgl_input_b[4][4];
static float16_t    rgl_output[4][4];
```

```c
void rgl_fill_matrix_a (void)
{
  //ROW 0 Fill
  rgl_input_a[0][0] = 1.00;
  rgl_input_a[0][1] = 2.00;
  rgl_input_a[0][2] = 3.00;
  rgl_input_a[0][3] = -4.00;

  //ROW 1 Fill
  rgl_input_a[1][0] = 5.00;
  rgl_input_a[1][1] = 6.00;
  rgl_input_a[1][2] = 7.00;
  rgl_input_a[1][3] = -8.00;

  //ROW 2 Fill
  rgl_input_a[2][0] = 9.00;
  rgl_input_a[2][1] = 10.00;
  rgl_input_a[2][2] = 11.00;
  rgl_input_a[2][3] = -12.00;
}
```

```c
void rgl_fill_matrix_b (void)
{
  //ROW 0 Fill
  rgl_input_b[0][0] = 1.00;
  rgl_input_b[0][1] = 5.00;
  rgl_input_b[0][2] = 9.00;

  //ROW 1 Fill
  rgl_input_b[1][0] = 2.00;
  rgl_input_b[1][1] = 6.00;
  rgl_input_b[1][2] = 10.00;

  //ROW 2 Fill
  rgl_input_b[2][0] = 3.00;
  rgl_input_b[2][1] = 7.00;
  rgl_input_b[2][2] = 11.00;

  //ROW 3 Fill
  rgl_input_b[3][0] = -4.00;
  rgl_input_b[3][1] = -8.00;
  rgl_input_b[3][2] = -12.00;
}
```

Demo

Matrix Multiply Example – Multiply & Print

```
void rgl_multiply_matrix (int num_rows_a, int num_cols_a, int num_rows_b, int num_cols_b)
{

    for (int i = 0; i < num_rows_a; i++)
    {
        for(int j = 0; j < num_cols_b; j++)
        {
            rgl_output[i][j] = 0;
            for(int k = 0; k < num_rows_b; k++)
            {
                rgl_output[i][j] += rgl_input_a[i][k] * rgl_input_b[k][j];
            }
        }
    }
}
```

```
void rgl_print_output_matrix (int num_rows, int num_cols)
{
  float16_t my_data;

    printf("Output of Matrix Multiply.\n");
    for (int r = 0; r < num_rows; r++)
    {
        for(int c = 0; c < num_cols; c++)
        {
            my_data = rgl_output[r][c];
            printf("(%6.2f),  ", my_data);
        }
        printf("\n");
    }
}
```


Demo

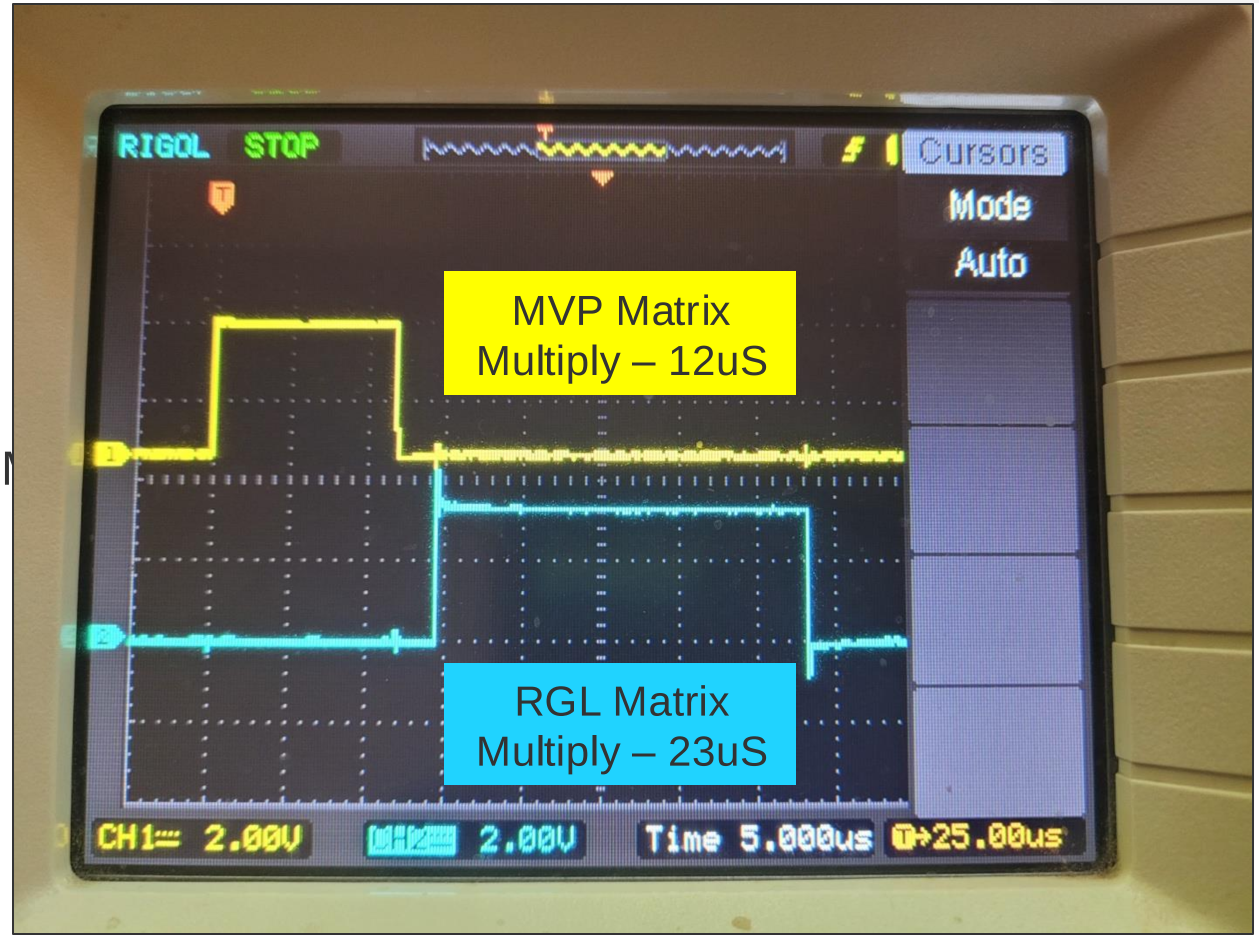
Matrix Multiply Example – Compare Functions

```c
void app_process_action(void)
{
  //printf("\n");
  //printf("Fill matrix A with values:\n");
  input_a[0] = 1.0;
  input_a[1] = 2.0;
  input_a[2] = 3.0;
  input_a[3] = -4.0;
  input_a[4] = 5.0;
  input_a[5] = 6.0;
  input_a[6] = 7.0;
  input_a[7] = -8.0;
  input_a[8] = 9.0;
  input_a[9] = 10.0;
  input_a[10] = 11.0;
  input_a[11] = -12.0;
  sl_math_matrix_init_f16(&matrix_a, 3, 4, input_a);
  //print_matrix(&matrix_a);

  //printf("Transpose matrix A into matrix B:\n");
  sl_math_matrix_init_f16(&matrix_b, 4, 3, input_b);
  sl_math_mvp_matrix_transpose_f16(&matrix_a, &matrix_b);
  //print_matrix(&matrix_b);

  //printf("Multiply matrix A with matrix B:\n");
  sl_math_matrix_init_f16(&matrix_z, 3, 3, output);
  GPIO_PinOutSet (gpioPortB , LED0);
  sl_math_mvp_matrix_mult_f16(&matrix_a, &matrix_b, &matrix_z);
  GPIO_PinOutClear (gpioPortB , LED0);
  //print_matrix(&matrix_z);

  //Test RGL matrix multiply
  rgl_fill_matrix_a();
  rgl_fill_matrix_b();
  GPIO_PinOutSet (gpioPortB , LED1);
  rgl_multiply_matrix(3, 4, 4, 3);
  GPIO_PinOutClear (gpioPortB , LED1);
  rgl_print_output_matrix (3, 3);
}
```

```
Output of Matrix Multiply.
( 30.00),  ( 70.00),  (110.00),
( 70.00),  (174.00),  (278.00),
(110.00),  (278.00),  (446.00),
```

Demo

Matrix Multiply – RGL vs MVP Performance



Thank You