Application Note

AC Arc Fault Detection Using Edge AI on MSPM0 Low-Power MCU

Akshat Aggarwal, Laavanaya Dhawan, Nathan Nohr, Vinamra Shrivastava

ABSTRACT

This application note demonstrates how to implement real-time AC arc fault detection using Edge AI on the low-cost MSPM0G5187 microcontroller with integrated Neural Processing Unit (NPU). The design combines the TIDA-010971 analog front end reference design with machine learning inference to detect series arc faults in residential and commercial electrical systems. Target performance includes less than 10ms response time and greater than 95% detection accuracy while maintaining immunity to masking loads per UL 1699 guidelines . This reference design enables engineers to develop Arc Fault Circuit Interrupter (AFCI) products for compliance with National Electrical Code (NEC) requirements.

Trademarks

Table of Contents

1 Introduction

The Arc Fault Problem

Arc faults are one of the leading causes of electrical fires in residential and commercial buildings. An arc fault occurs when electrical current flows through an unintended path, creating a high-temperature plasma arc that can ignite surrounding materials. According to the U.S. Consumer Product Safety Commission (CPSC) and National Fire Protection Association (NFPA), arc faults cause over 30,000 home fires annually, resulting in hundreds of deaths and over $1 billion in property damage. (Source: NFPA Electrical Fire Statistics, 2019-2023)

Table 1-1. Types of Arc Faults

Type	Description	Characteristics
Series Arc	High-impedance connection in series with the load (for example, damaged wire, loose connection)	Current flows through arc; load still operates but with reduced power
Parallel Arc	Line-to-neutral or line-to-ground fault through damaged insulation	High fault current; can occur without load connected

Comparison of series and parallel arc faults. Series arcs occur in the current path with load in circuit; parallel arcs occur across line and neutral or ground

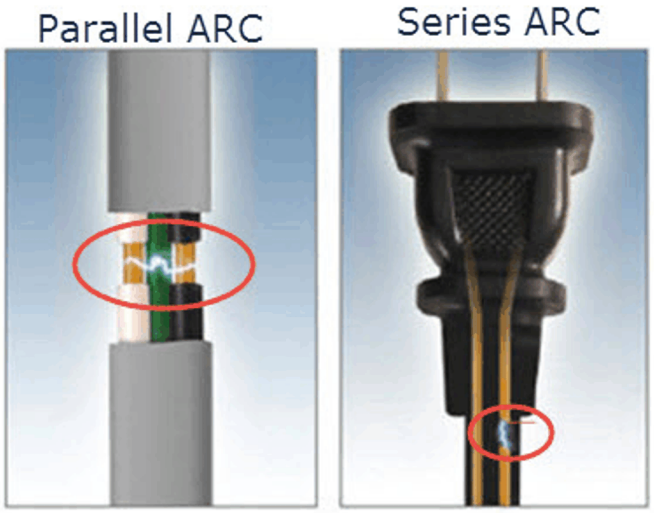


Figure 1-1. Series and Parallel Arc Faults

Why Traditional Protection Fails

Traditional circuit breakers and fuses protect against overcurrent conditions but cannot detect arc faults because:

- Series arcs may draw less current than the normal load
- Arc current waveforms differ from simple overcurrent signatures
- Arcing occurs at frequencies (kHz to MHz) not monitored by thermal breakers

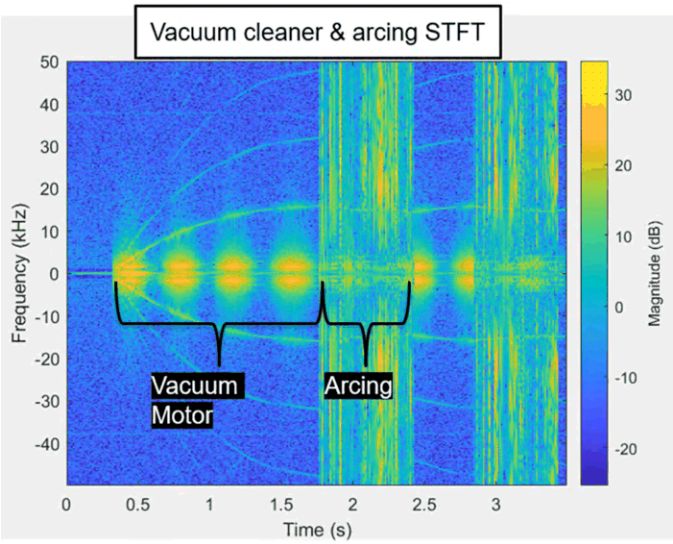


Figure 1-2. Arc Fault vs Normal Load Waveforms

UL 1699 Standard Overview

UL 1699 is the safety standard for Arc-Fault Circuit-Interrupters (AFCIs) that defines testing requirements for residential circuit protection devices. Compliance with UL 1699 is required for AFCIs sold in North America, and the National Electrical Code (NEC) mandates AFCI protection in most living spaces.

Table 1-2. Key UL 1699 Test Categories

Test Category	Description	ML Model Relevance
Carbonized Path Arc	Arc through carbonized insulation	Primary detection target
Point Contact Arc	Metal-to-metal intermittent contact	High-frequency signature detection
Masking Tests	Normal loads mimicking arc signatures	False positive prevention
Unwanted Tripping	Dimmers, vacuums, drills, SMPS	Model must NOT trip on these

This application note provides a reference design to help engineers develop AFCI products. Production devices require formal UL certification testing. The design principles and test methodologies presented align with UL 1699 requirements.

Table 1-3. Design Targets vs. UL 1699

Metric	UL 1699 Requirement	This Design Target
Series Arc Detection	Required	Yes
Parallel Arc Detection	Required	No
Response Time	Varies by test	<10ms
Masking Load Immunity	Must not nuisance trip	>99% specificity

Why Edge AI for Arc Fault Detection?

Traditional arc fault detection relies on analog circuits that monitor specific frequency bands or waveform characteristics. While effective, these approaches have limitations:

- Fixed thresholds cannot adapt to varying load conditions
- Limited feature extraction misses complex arc signatures
- High false positive rates with challenging loads (dimmers, motors)

Edge AI enables a fundamentally different approach:

1. Complex Pattern Recognition: Neural networks can learn subtle differences between arc signatures and normal load noise
2. Multi-Feature Analysis: Combine time-domain, frequency-domain, and energy features
3. Adaptive Detection: Models trained on diverse datasets generalize better than fixed rules
4. On-Device Inference: Real-time detection without cloud connectivity

Table 1-4. Edge AI vs. Traditional Analog AFCI Comparison

Aspect	Traditional Analog AFCI	Edge AI AFCI (This Design)
Detection Method	Fixed analog filters + comparators	ML model with learned features
Feature Extraction	Hardware filters (limited bands)	Software FFT (full spectrum analysis)
Threshold Adaptation	Fixed or manually tuned	Learned from training data
False Positive Handling	Additional analog filters	ML model trained on masking loads
Update Capability	Hardware change required	Firmware/model update possible
Development Effort	Analog circuit design expertise	ML training + embedded deployment
BOM Cost	Multiple analog ICs	Single MCU with NPU
Power Consumption	Varies (typically higher)	Low Power MCU
Flexibility	Application-specific	Adaptable to different scenarios

Why MSPM0?

The MSPM0G5187 is uniquely suited for Edge AI arc fault detection:

Table 1-5. MSPM0G5187 Features

Feature	Benefit for Arc Detection
TinyEngine NPU	Hardware-accelerated neural network inference
80MHz Cortex-M0+	Fast DSP for real-time feature extraction
12-bit 1.6 Msps ADC	High-speed sampling captures arc signatures
128 KB Flash	Sufficient for ML model and application code
32 KB SRAM	Adequate tensor arena for inference
Low Power	Always-on monitoring with low power MCU
Cost Effective	Low BOM cost for consumer AFCI products
Small Package	4mm × 4mm options for compact designs

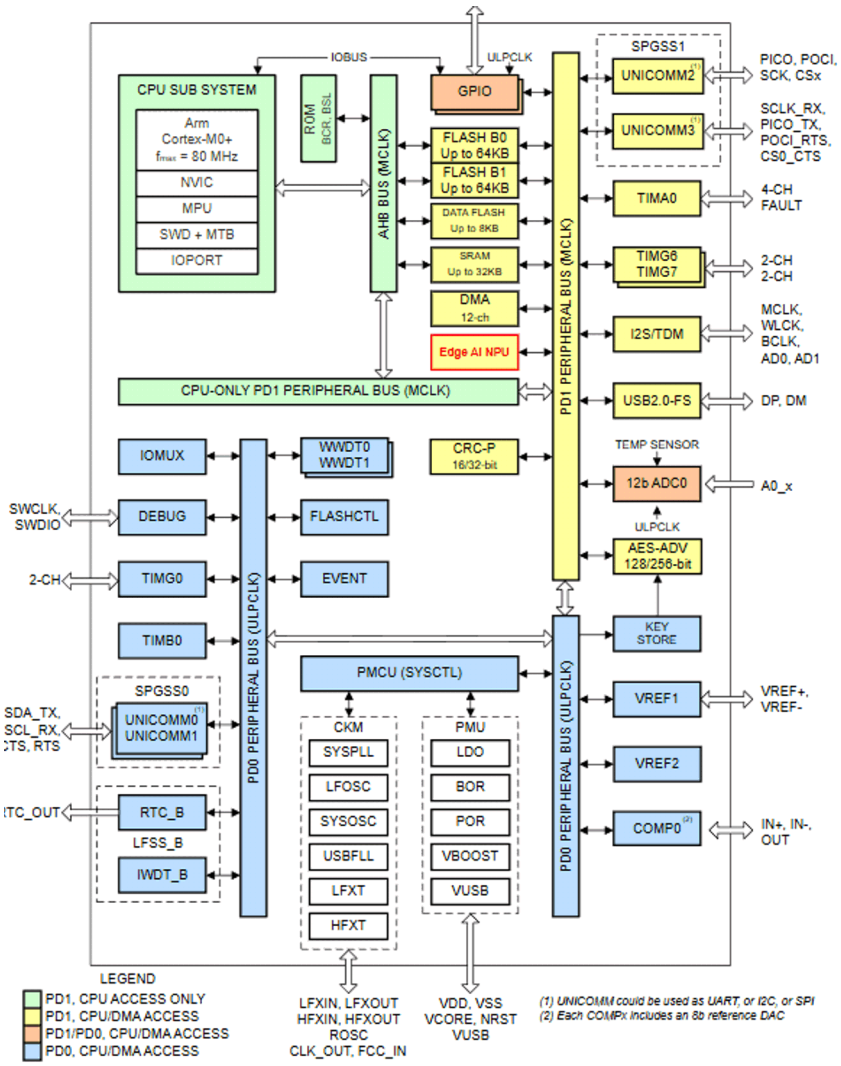


Figure 1-3. MSPM0G5187 Functional Block Diagram

Note

Other MSPM0 variants without NPU can run Edge AI inference using software-only implementations but without hardware acceleration.

2 Quick Start Guide

This section provides the fastest path to running the arc fault detection demo. For detailed explanations, refer to subsequent sections.

Requirements

What You Need

Hardware:

- LP-MSPM0G5187 LaunchPad
- TIDA-010971 Analog Front End Board
- USB Type-C cable
- USB isolator (required for AC testing)

Software:

- Code Composer Studio 12.x or later
- MSPM0 SDK 2.08.00 or later

Quick Start Steps

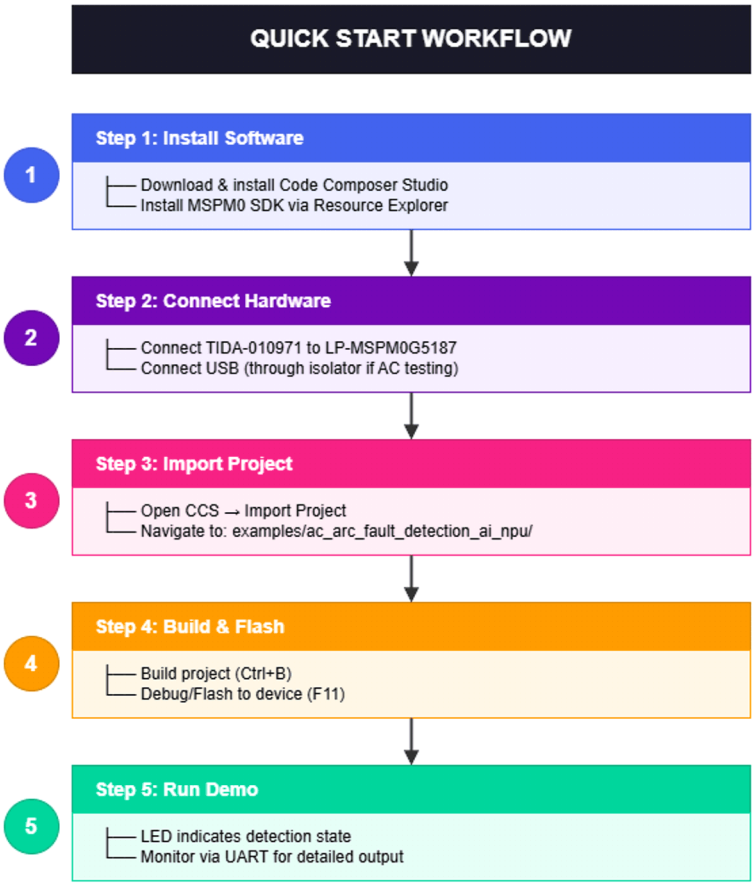


Figure 2-1. Quick Start Workflow Showing the Five Main Steps from Software Installation to Running the Demo

Step-by-Step Instructions

Step 1: Install Software

- Download Code Composer Studio from ti.com/ccs
- During installation, select "MSPM0 Microcontrollers"
- Launch CCS and open Resource Explorer (View → Resource Explorer)
- Search for "MSPM0 SDK" and install version 2.10.00 or later

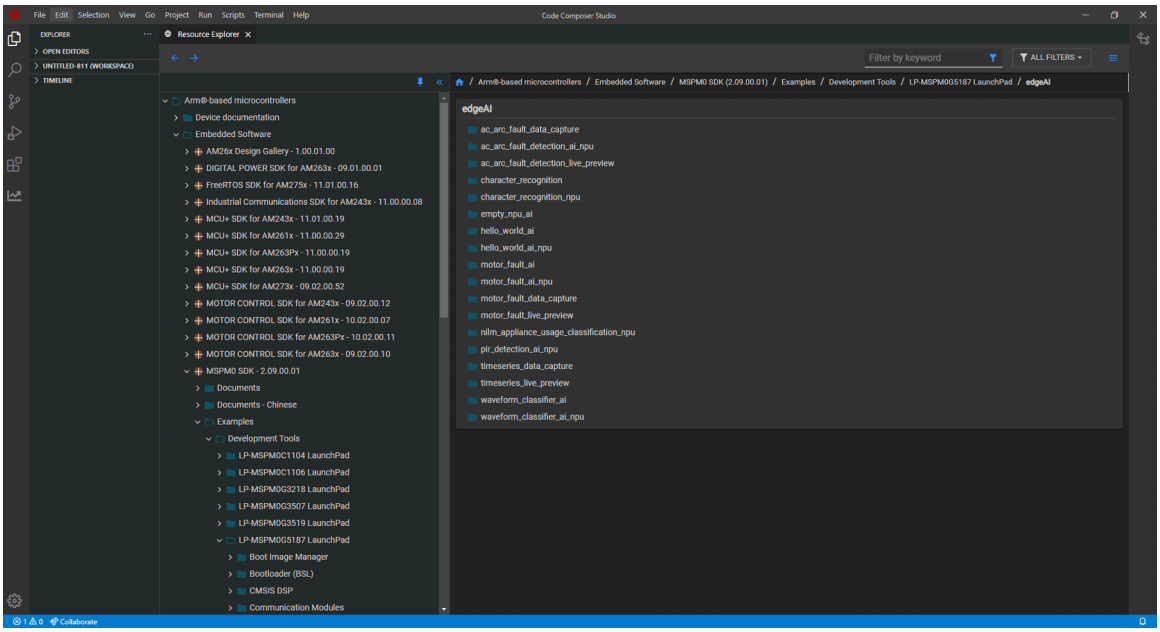


Figure 2-2. Code Composer Studio Resource Explorer showing MSPM0 SDK installation

Step 2: Connect Hardware

- Connect the TIDA-010971 board to the LP-MSPM0G5187 LaunchPad headers
- Verify pin alignment (see Section 3.1 for pinout)
- Connect USB Type-C cable to LaunchPad

WARNING

When testing with AC mains voltage, always use a USB isolator between the LaunchPad and the computer. The TIDA-010971 operates at hazardous voltages.

TIDA-010971 board connected to LP-MSPM0G5187 LaunchPad. Note proper header alignment and USB isolator connection.

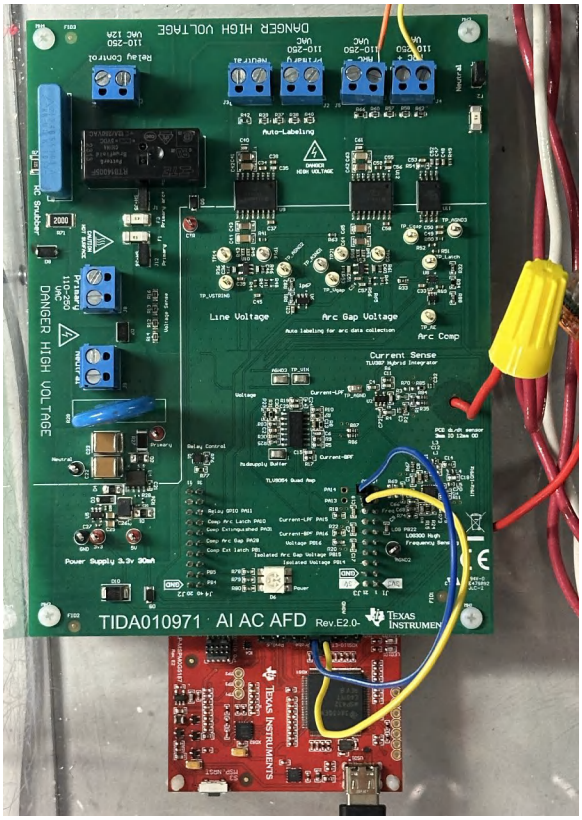


Figure 2-3. TIDA-010971 Board Connected to LP-MSPM0G5187 LaunchPad

Step 3: Import Project

- In CCS, select File → Import → CCS Projects
- Browse to SDK installation: <SDK_PATH>/examples/ac_arc_fault_detection_ai_npu/LP_MSPM0G5187/
- Select the project and click Finish

Step 4: Build and Flash

- Right-click project → Build Project (or Ctrl+B)
- Verify build completes without errors
- Click Debug (F11) to flash and start debugging
- Click Resume (F8) to run the application

Step 5: Verify Operation

Table 2-1. LED Status Table

LED State	Meaning
Red ON	Arc fault detected
Both OFF	System running, no arc detected

Next Steps

After running the demo:

Table 2-2. Next Steps

Goal	Section
Train your own model	[Section 6: Model Development](#6-model-development)
Modify the firmware	[Section 7: Firmware Implementation](#7-firmware-implementation)
Optimize power consumption	[Section 10: Power Optimization](#10-power-optimization)
Debug issues	[Section 12: Troubleshooting](#12-troubleshooting)

3 Technical Background

Arc Fault Electrical Characteristics

Arc faults produce distinctive electrical signatures that differ from normal load operation:

Time-Domain Characteristics:

- Current discontinuities at zero crossings (arc extinguishes when voltage drops)
- Irregular current waveform with high-frequency noise superimposed
- Current magnitude may be lower than normal load (series arcs)
- Shoulder phenomenon: current *shoulder* near zero crossing as arc re-ignites

Frequency-Domain Characteristics:

- Broadband noise from DC to tens of MHz
- Time-varying pink noise spectrum (1/f roll-off)
- Arc energy present across wide frequency range
- Distinct spectral signature compared to normal switching transients

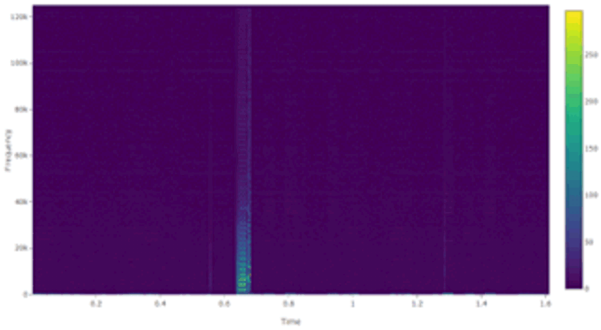


Figure 3-1. Normal Frequency Spectrum

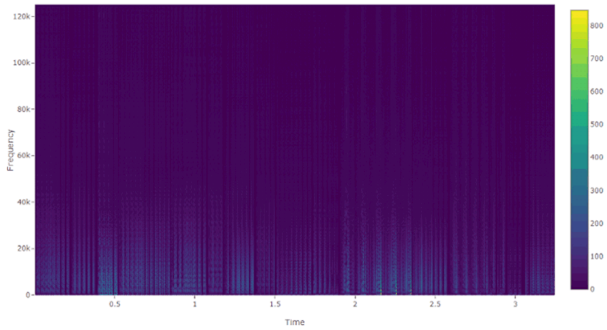


Figure 3-2. Arc Fault Frequency Spectrum

Table 3-1. Arc Signature vs Load Noise

Characteristic	Arc Fault	Normal Load (Motor/Dimmer)
Temporal Pattern	Random, chaotic	Periodic, predictable
Zero-Crossing Behavior	Extinguishes/re-ignites	Smooth transitions
Energy Distribution	Uniform across spectrum	Concentrated at harmonics

Signal Processing Overview (TIDA-010971)

The TIDA-010971 implements a multichannel sensing strategy using a PCB Rogowski coil with >40 MHz bandwidth:

Table 3-2. Signal Channels

Channel	Frequency Range	Purpose	ML Feature Usage
Band-pass current	5-50kHz	Arc noise in audible/ultrasonic range	Primary input for FFT features
LOG300 HF output	1-10MHz	High-frequency arc energy	Arc energy magnitude
Low-pass current	DC-60Hz	Fundamental current magnitude	Load level detection
Line voltage	60 Hz	Zero-crossing reference	Phase synchronization

Key Design Concepts

For complete circuit schematics, component values, and design calculations, refer to the TIDA-010971 Design Guide.

- PCB Rogowski Coil: Differential coil outputs di/dt signal; self-resonance >40MHz enables wideband arc detection
- Hybrid Integrator: TLV387-based circuit converts di/dt to current-proportional signal with 68.6 dB gain
- LOG300 Detector: Provides 98 dB dynamic range for MHz-frequency arc energy detection

Feature Extraction Pipeline

The firmware extracts frequency-domain features from raw ADC samples:

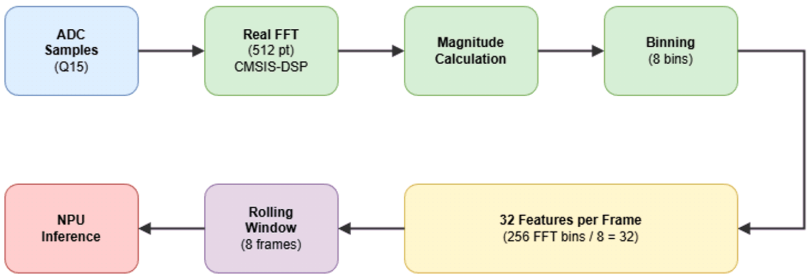


Figure 3-3. Feature Extraction Pipeline

Timing Relationships

- ADC sampling interval: 9.3µs (1/107 kHz approximately 9.3µs)
- Frame period: 512 samples × 9.3µs equals approximately 4.76ms
- Rolling window: 8 frames × 4.76ms equals approximately 38ms of temporal context
- Inference rate: Once per frame (approximately 4.76ms)

Configuration Parameters (user_input_config.h):

Note

Note on *Rolling Window*: The term "window" here refers to the temporal sliding window of eight consecutive frames, not an FFT window function (for example, Hanning, Hamming). Each new frame shifts the window forward, providing continuous temporal context for arc detection.

```
#define FE_FRAME_SIZE 512              // ADC samples per frame
#define FE_FEATURE_SIZE_PER_FRAME 32   // Output features per frame
#define FE_NUM_FRAME_CONCAT 8          // Frames in rolling window
#define FE_BIN_SIZE 8                  // FFT bins per feature bin
#define FE_BIN_OFFSET 1                // Skip DC component
#define FE_COMPLEX_MAG_SCALE_FACTOR 5 // Magnitude scaling
```

Processing Steps:

- Real FFT: 512-point FFT using ARM CMSIS-DSP (arm_rfft_q15)
- Complex Magnitude: Calculate magnitude of each frequency bin
- DC Removal: Skip DC component (FE_BIN_OFFSET = 1)
- Binning: Average 8 adjacent FFT bins → 32 features
- Normalization: Scale to INT8 range (-128 to 127)
- Frame Concatenation: Stack eight frames in rolling window → 256 total features

Processing Steps

ML Model Architecture

To support a wide range of memory, latency, and accuracy requirements on MSPM0-class devices, the Arc Fault Detection solution provides a family of lightweight convolutional neural network (CNN) models. These models share a common 3-layer convolutional backbone and differ only in channel depth, allowing scalable deployment from ultra-low-footprint to higher-accuracy configurations. The neural network classifies the extracted features as arc fault or normal.

Table 3-3. Model Architecture Comparison

NOTE: The performance metrics below are measured for an input dimension of N=1, C=1, H=256, W=1

Model Name	Parameter	Flash Size (B)	SRAM Size (B)	Inference Time	Accuracy	Recommendation
ArcFault_model_200_t	~200	3652	900	197.82us	99.60%	Simplest, smallest and fastest model.
ArcFault_model_300_t	~300	3864	1644	246.91 us	99.60%	Faster than the 700 and 1400 variant, but also handles less complex data.
ArcFault_model_700_t	~800	4496	1644	288.36 us	99.42%	Large model, sweet spot between inference speed and memory occupied.
ArcFault_model_1400_t	~1600	5564	2408	713.31 us	99.88%	Most accurate model, use for complex data scenarios (Recommended)

The Arc Fault models mentioned above are proprietary and hence are only available in the GUI Based EdgeAI Studio.

Recommended 1D CNN architecture for arc fault detection (Arc Fault Model 1400t model)

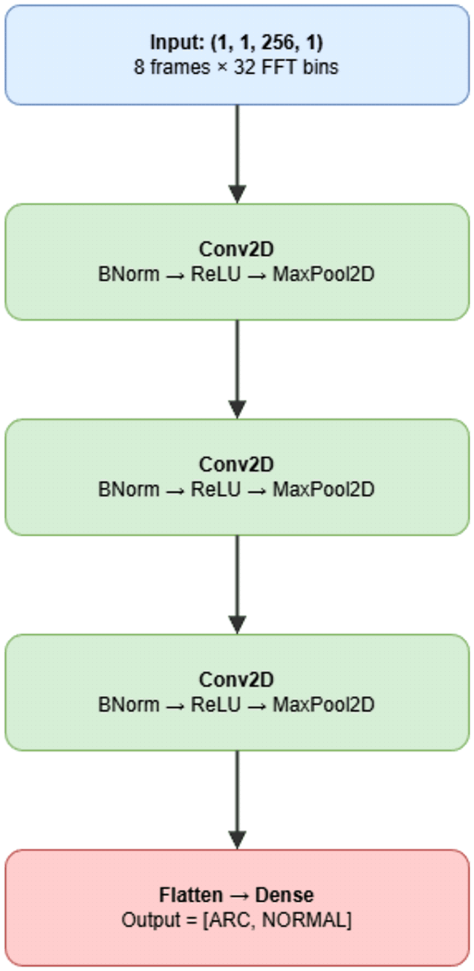


Figure 3-4. Recommended 1D CNN Architecture for Arc Fault Detection

NPU Acceleration

The MSPM0G5187 includes an integrated TinyEngine™ NPU that accelerates neural network inference:

NPU Features:

- Hardware acceleration for common NN operations
- Instruction and parameter memory for model storage
- Interrupt-driven completion signaling
- TVM runtime integration via TI Neural Network Compiler (TI-NNC)

Integration with TI-NNC

The TI Neural Network Compiler generates:

- model.a: Compiled model library
- tvmgen_default.h: C interface header
- user_input_config.h: Feature extraction configuration

4 System Requirements

Hardware Requirements

Table 4-1. Hardware Requirements

Component	Part Number	Description
MCU/LaunchPad	LP-MSPM0G5187	MSPM0G5187 LaunchPad Development Kit
USB Cable	USB Type-C	For programming and debug
USB Isolator	Any USB 2.0 isolator	Required for safety during AC testing

Primary Components

TIDA-010971 Specifications

Table 4-2. TIDA-010971 Specifications

Parameter	Value
Power Draw	7-17 mA
Rogowski Bandwidth	\50 Hz to 40 MHz
Rogowski Sensitivity	~0.4 mV/A at 6 kHz
Band-Pass Filter	5 kHz to 50 kHz
LOG300 HF Detection	1 MHz to 10 MHz
Maximum Current Sensing	320 A (without clipping)

LaunchPad Pin Connections

Table 4-3. LaunchPad Pin Connections

Pin	ADC Channel	Signal Description
PA15	ADC0_15	Low-pass filtered current (60 Hz fundamental)
PA21	ADC0_24	LOG300 output (HF arc energy)
PB14	ADC0_21	Isolated line voltage (AMC3330)
PA31	GPIO	ARC Indicator Pin (set-ARC, not set – No ARC)
PA11	GPIO	Relay control output
PB4	GPIO	Red LED indicator
PB5	GPIO	Green LED indicator

TIDA-010971 Key Components

The TIDA-010971 uses specialized analog components including the TLV387 zero-drift op-amp for the hybrid integrator, TLV9054 quad op-amp for band-pass and low-pass filtering, LOG300 high-frequency log detector (98 dB dynamic range), AMC3330 isolated amplifier, and UCC28881 offline buck converter for power supply.

For complete IC specifications, circuit schematics, and design files, refer to the TIDA-010971 Design Guide

Software Requirements

Table 4-4. Software Requirements

Software	Version	Purpose
MSPM0 SDK	2.10.00 or later	Device drivers and libraries
TI Edge AI Studio	1.6.0 or later	Model training and deployment GUI tool
Model Maker	R1.3 or later	Model training and deployment CLI tool
ARM CMSIS-DSP	Included in SDK	FFT and DSP functions
Python	3.10 or later	Model training/data labelling scripts

Model Requirements

Table 4-5. Model Requirements

Parameter	Specification
Input Dimension	8 frames × 32 FFT bins = 256 total features
Output Shape	(1, 2) - [arc_confidence, normal_confidence]
Quantization	INT8 (mandatory - no FPU on Cortex-M0+)
Model Size	Approximately 14KB parameters
Tensor Arena	Approximately 8 KB SRAM

Tensor Shape Explanation

- batch (1): Single inference at a time
- channels (1): Single input channel (grayscale-like representation)
- height (1024): Concatenated feature vector (8 frames × 32 features)
- width (1): Single element width (vector representation as 2D)

5 Hardware Design

System Overview

The complete arc fault detection system consists of the TIDA-010971 analog front end connected to the LP-MSPM0G5187 LaunchPad:

Complete system block diagram showing TIDA-010971 analog front end connected to LP-MSPM0G5187 for Edge AI arc fault detection

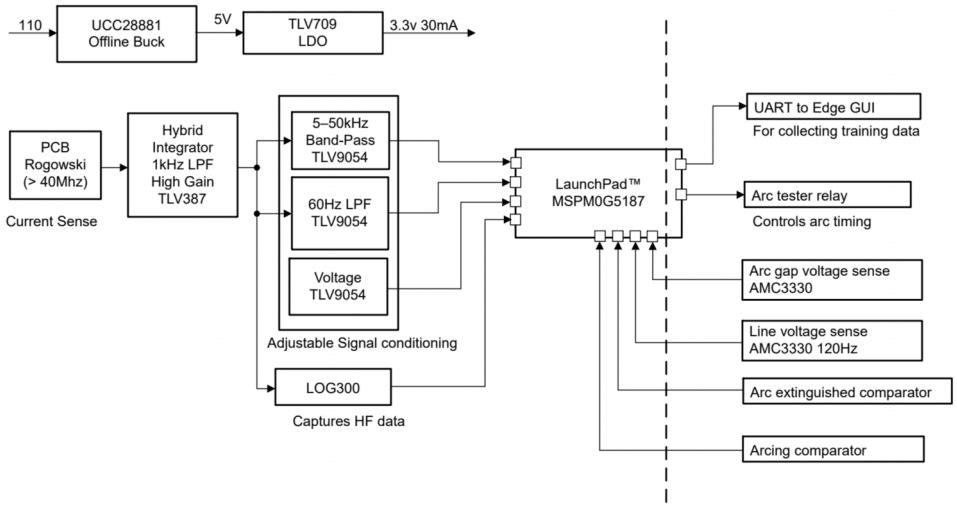


Figure 5-1. TIDA-010971 System Block Diagram

Key Components

- TIDA-010971: Provides analog signal conditioning, power supply, and isolation
- LP-MSPM0G5187: Performs ADC sampling, feature extraction, NPU inference, and trip decision

For complete system schematics, PCB layout, and design files, refer to the TIDA-010971 Design Guide.

Hardware Setup

Board Connections

Connect the TIDA-010971 to the LP-MSPM0G5187 using the BoosterPack headers. Verify pin alignment before powering on. Connect the Arc generator setup with the analog front end for generating different arc or normal scenarios.

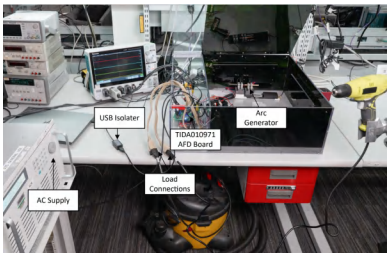


Figure 5-2. Arc Generator Test Setup

Safety Considerations

WARNING

Hazardous Voltages

This design operates at AC mains voltages (110-250 VAC) that can cause severe injury or death. Follow ALL safety guidelines:

1. USB Isolation Required: ALWAYS use a USB isolator between the LaunchPad and your PC during AC testing
2. Enclosure: Operate only in a properly grounded, insulated enclosure
3. Current Limiting: Use a current-limited AC source during development
4. No-Touch Rule: NEVER touch the board or connections while energized
5. Discharge Time: Allow capacitors to discharge (>5 minutes) before handling
6. Qualified Personnel: Only qualified electrical engineers should perform AC testing

6 Model Development

This section provides a guide to developing and deploying an arc fault detection ML model using TI's Edge AI tools.

Development Workflow Overview

End-to-end model development workflow from data collection through deployment

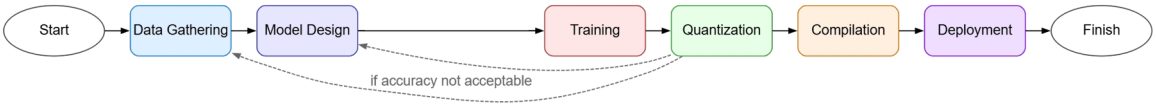


Figure 6-1. Model Development Workflow

Data Collection

Hardware Setup

- Flash the data capture firmware: examples/ac_arc_fault_data_capture/LP_MSPM0G5187/
- Connect TIDA-010971 to LP-MSPM0G5187 via BoosterPack headers
- Connect USB (through isolator for AC testing)
- Serial connection: 5,820,000 baud, binary frames

Data Collection Scenarios

Collect data for each class under controlled conditions. Each load type contains 20000+ half-cycle captures minimum (approximately three minutes split evenly across normal and arcing configurations).

Table 6-1. Class 1: Normal Operation (No Trip)

Load Type	Description	Captures
Inductive	AC relay, compressor motor, vacuum, shop tool	10000+
LED	Halogen lighting, 600W-1000W, with dimmer	10000+
SMPS	600W AC/DC	10000+
Mixed	Combination of above with 5A resistive load	10000+

Table 6-2. Class 2: Arc Fault Masking Loads

Arc Type	Description	Captures
Configuration A	Load with arc generator in series	10000+
Configuration B	Load with arc generator in series and parallel 5A resistive load	10000+
Configuration C	Load in parallel to 5A resistive load with arc generator	10000+
Configuration D	Load and 5A resistive load both with arc generator	10000+

Data Labeling

The captured data is labeled to two classes. The labeling tool provides two configurable modes for generating ARC and NORMAL training data from raw waveform CSVs: Frame Mode and Window Mode. Both modes use arc-voltage–based heuristics to determine ground-truth labels and are independent of any machine-learning model. While both modes are supported to accommodate different datasets and front-end architectures, Frame Mode is the recommended default for most applications due to the ability to better preserve event continuity and produce cleaner class boundaries.

Frame Mode (Recommended)

Purpose: Frame Mode is designed to capture natural arc event durations and produce variable-length ARC and NORMAL segments that closely reflect real electrical behaviour. It is particularly suitable for LOG300-based datasets and scenarios where preserving temporal continuity is important.

How it Works:

1. The waveform is divided into fixed-length frames
2. For each frame, arc-gap voltage statistics (e.g., peak and RMS) are evaluated against configurable threshold
3. Temporal smoothing is applied across consecutive frames to suppress isolated spikes and switching transients. A state-machine–based merge combines consecutive ARC or NORMAL frames into longer segments. Segments below a minimum duration are discarded to avoid training on short, noisy fragments. Frames overlapping confirmed ARC regions are excluded from NORMAL output to verify *pure normal* samples

Output Characteristics

1. Variable-length ARC and NORMAL segments
2. Strong noise suppression and reduced label jitter
3. Preserves the natural start/stop boundaries of arc events
4. Configurable saved channel (e.g., LOG300 envelope or current), while labeling logic always uses voltage.

When to Use:

1. LOG300 or envelope-based front ends
2. Applications requiring high label fidelity and event continuity
3. Dataset creation for models sensitive to transient noise or boundary artifacts

Window Mode (Recommended)

Purpose: Window Mode generates fixed-length training samples using a sliding-window approach. It is useful when uniform input sizes are required or when building datasets for architectures that expect constant-length features.

How it Works:

1. The signal is first divided into frames.
2. A window of N consecutive frames slides across the waveform.
3. Within each window, the number of arc-like frames (based on voltage thresholds) is counted.
4. If the count exceeds a configurable threshold, the entire window is labeled ARC; if voltage levels are sufficiently low and stable, it is labeled NORMAL; otherwise, the window is discarded.
5. The extracted waveform corresponding to the labeled window is saved as a fixed-size sample.

Output Characteristics:

1. Fixed-length ARC and NORMAL windows
2. Easier batching and consistent model input dimensions
3. May discard ambiguous windows to maintain confidence
4. Labeling remains voltage-driven; saved channel is typically current

When to Use

1. Pipelines requiring uniform sample lengths.
2. Scenarios where strict batching simplicity outweighs event-continuity requirements.

Recommendation

Both modes are fully supported and share the same voltage-based ground-truth logic. However, Frame Mode is recommended as the default because it: - better preserves real-world arc event continuity, - reduces noise-induced false labels through temporal smoothing, and - produces cleaner NORMAL samples by explicitly excluding arc-adjacent regions. Window Mode remains valuable where fixed-size inputs or rapid dataset standardization are primary requirements.

For more information refer to Tinyml-modelzoo->examples->ac_arc_fault->readme.md

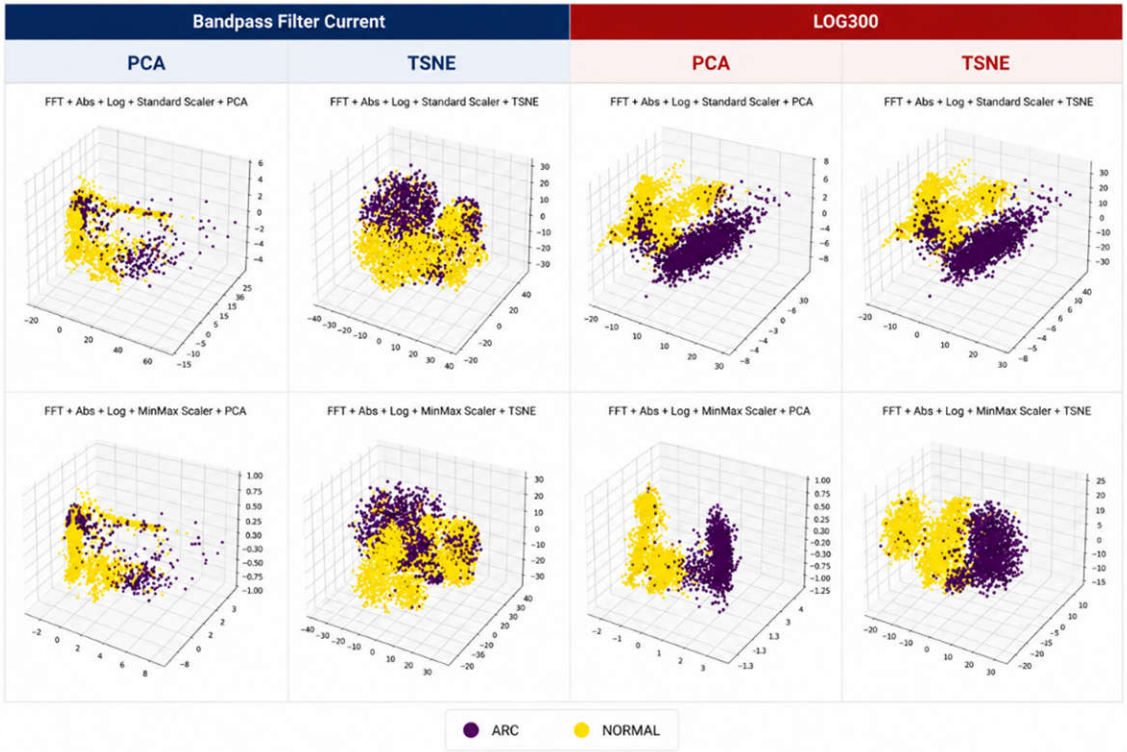
Dataset Analysis and Selection

The Goodness of Fit (GOF) plots were used as an early-stage assessment to evaluate feature separability in the dataset before model training. These plots helped compare the distribution of ARC and NORMAL samples across different preprocessing combinations, including FFT, wavelet transform, scaling, PCA, and t-SNE. Strong class separation in the GOF plots indicates that the selected signal chain and feature extraction method are suitable for classification, while overlap between classes suggests that further tuning may be required.

For the bandpass-filtered current dataset, the GOF plots showed noticeable ARC/NORMAL overlap, particularly with FFT-based preprocessing, indicating that some arc-related features were being masked. In contrast, the LOG300-based dataset produced clearer class clusters, suggesting improved feature separability and better suitability for downstream model training. Additional details on dataset goodness of fit, feature extraction quality,

GOF graphs, and file-level classification summaries are available in the TI Tiny ML Tensorlab User Guide under the Advanced Features.

GOF comparison of bandpass-filtered current & LOG300 datasets



Model Architecture

While the Arc Fault models mentioned in section 4.4 are the recommended choice of models for AC Arc Fault detection. A wide range of generic timeseries models can also be employed for classification.

Generic time-series model templates are available in both TI Edge AI Studio (GUI) and TensorLab (CLI) to enable rapid development and evaluation of embedded ML applications. These templates provide ready-to-use, NPU-compatible architectures that can be trained on custom datasets with minimal configuration effort.

One such reference architecture is the CLS_6K_NPU model, a lightweight depthwise-separable convolutional neural network. This model, along with a range of other reference classification architectures, is available within TI Edge AI Studio (GUI) and the Model Zoo provided through the CLI tools.

```
import torch
import torch.nn as nn
import torch.nn.functional as F


class CLS_6K_NPU(nn.Module):
def __init__(self, input_features=512, variables=1, num_classes=2):
super(CLS_6K_NPU, self).__init__()

self.bn0 = nn.BatchNorm2d(variables)

self.conv1 = nn.Conv2d(variables, 16, kernel_size=(3, 1), stride=(4, 1), padding=0, bias=False)
self.bn1 = nn.BatchNorm2d(16)

self.pool1 = nn.MaxPool2d(kernel_size=(3, 1), stride=(2, 1))self.dw3 = nn.Conv2d(16, 16,
kernel_size=(3, 1), stride=(1, 1), padding=0,
groups=16, bias=False)
self.bn3a = nn.BatchNorm2d(16)

self.pw3 = nn.Conv2d(16, 32, kernel_size=(1, 1), stride=(1, 1), padding=0, bias=False)
self.bn3b = nn.BatchNorm2d(32)
```

```
self.pool2 = nn.MaxPool2d(kernel_size=(3, 1), stride=(2, 1))

self.dw5 = nn.Conv2d(32, 32, kernel_size=(3, 1), stride=(1, 1), padding=0,
groups=32, bias=False)
self.bn5a = nn.BatchNorm2d(32)

self.pw5 = nn.Conv2d(32, 48, kernel_size=(1, 1), stride=(1, 1), padding=0, bias=False)
self.bn5b = nn.BatchNorm2d(48)

self.conv6 = nn.Conv2d(48, 16, kernel_size=(5, 1), stride=(1, 1), padding=0, bias=False)
self.bn6 = nn.BatchNorm2d(16)

self.gap = nn.AdaptiveAvgPool2d(output_size=(4, 1))
self.fc = nn.Linear(64, num_classes)

def forward(self, x):

x = self.bn0(x)
x = self.conv1(x)
x = F.relu(self.bn1(x))
x = self.pool1(x)
x = self.dw3(x)
x = F.relu(self.bn3a(x))
x = self.pw3(x)
x = F.relu(self.bn3b(x))

x = self.pool2(x)
x = self.dw5(x)
x = F.relu(self.bn5a(x))
x = self.pw5(x)
x = F.relu(self.bn5b(x))
x = self.conv6(x)
x = F.relu(self.bn6(x))
x = self.gap(x) # (B, 16, 4, 1)
x = x.view(x.size(0), -1) # (B, 64)
x = self.fc(x) # (B, num_classes)
return x
```

Note

This example uses TI Edge AI Studio GUI/CLI tools which supports PyTorch framework

Model Training

Table 6-3. Recommended Hyperparameters for ArcFault_model_1400_t:

Parameters	Value
Batch size	50
Learning rate	0.04
Optimizer	SGD
Weight decay	1e-5
Dataset split	60% train, 30% val, 10% test

Training Options

- GUI: Use **CCStudio™ Edge AI Studio GUI** for guided training workflow
- CLI: Use tinyml-tensorlab for command-line training

Enable Quantize-Aware Training (QAT) for best INT8 accuracy. Tinyml-tensorlab and Edge AI Studio have inbuilt support for 8 bit NPU aware quantization.

Quantization

Why INT8 Quantization?

The MSPM0G5187 Cortex-M0+ has no hardware floating-point unit:

- FP32 operations require software emulation (very slow)
- INT8 operations execute in single cycles
- NPU only supports INT8 operations

- ~4x reduction in model size

Quantize-Aware Training (QAT)

QAT simulates quantization during training for best accuracy preservation (typically <1% drop):

```
import torch.quantization as quant

# Prepare model for QAT
model.qconfig = quant.get_default_qat_qconfig('fbgemm')
quant.prepare_qat(model, inplace=True)

# Train with simulated quantization
for epoch in range(num_epochs):
    train_one_epoch(model, train_loader)

# Convert to fully quantized model
model_int8 = quant.convert(model, inplace=False)
```

Input Normalization

The normalization parameters are model-specific and generated by TI-NNC:

```
// From user_input_config.h - [MODEL_SPECIFIC]
#define FE_NORM_BIAS    [MODEL_SPECIFIC]  // Generated by TI-NNC
#define FE_NORM_SCALE   [MODEL_SPECIFIC]  // Generated by TI-NNC
#define FE_NORM_SHIFT   [MODEL_SPECIFIC]  // Generated by TI-NNC
```

Important: Always use the values from your model's user_input_config.h file generated during compilation.

Model Compilation with TI-NNC

The TI Neural Network Compiler (TI-NNC) converts your trained “onnx” INT8 model for MSPM0 NPU execution.

Compilation generates:

- model.a - Compiled model library (~14 KB)
- tvmgen_default.h - C interface header
- user_input_config.h - Feature extraction configuration

Integration into CCS Project:

- Copy generated files to project/model/ directory
- Add model.a to linker input files
- Include headers in source: #include *model/tvmgen_default.h*

For installation, CLI syntax, and detailed compilation instructions, see the TI-NNC User's Guide.

7 Firmware Implementation

This section details the firmware architecture for real-time arc fault detection on the LP-MSPM0G5187.

System Architecture

Main Loop State Machine:

Firmware state machine diagram showing the main application loop and interrupt handlers

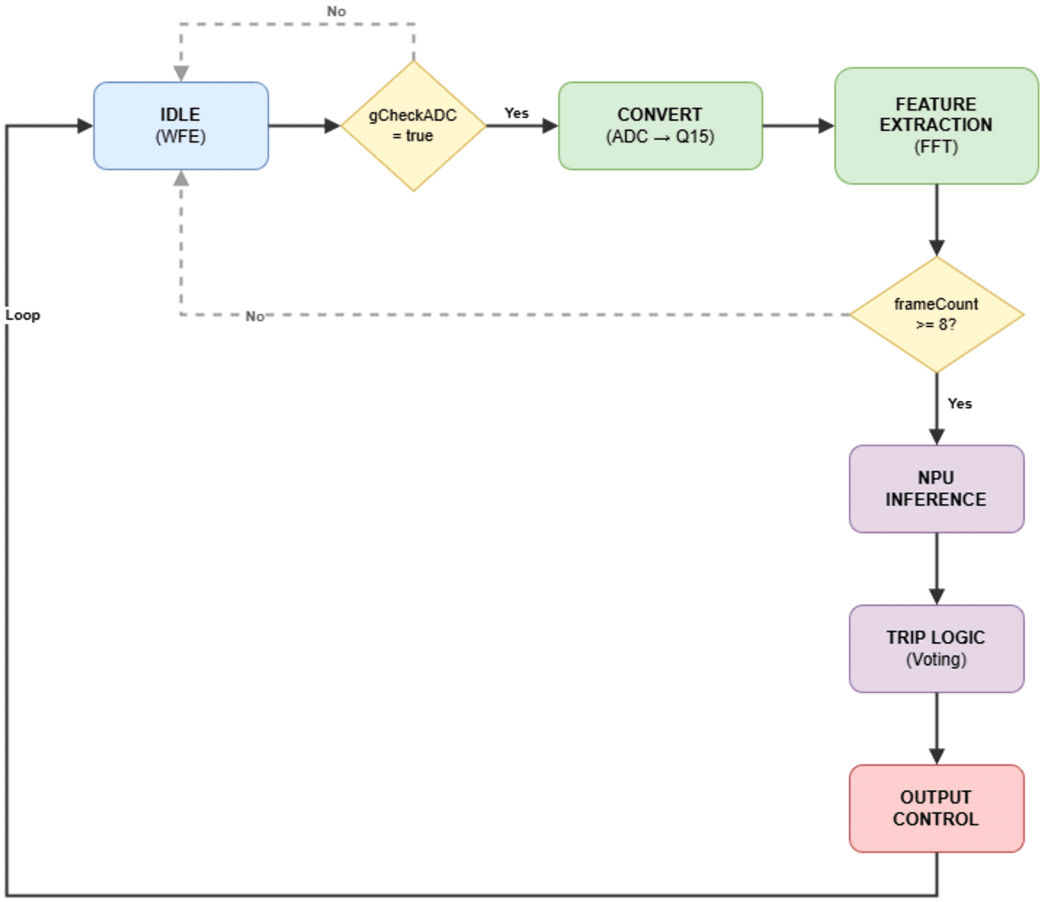


Figure 7-1. Firmware State Machine Diagram

Peripheral Configuration

Configure the MSPM0G5187 peripherals using TI SysConfig. Key settings:

Table 7-1. Peripheral Configuration

Peripheral	Configuration
ADC12_0	12-bit, SYSOSC clock, DMA enabled, Channel ADC0_14
DMA	Ping-pong transfer, 512 samples per buffer
GPIO	PA11 (relay), PB4/PB5 (LEDs), PA31 (comparator input)

Memory Map

Table 7-2. Memory Map

Flash Memory (128 KB)	SRAM (32KB)
Available (approximately 94 KB)	Available (approximately 23 KB)
Model (approximately 5.5 KB)	Stack (approximately 2 KB)
Model (approximately 5.5 KB)	Tensor Arena (2 KB)
App Code (approximately 20 KB)	Feature Buf (1 KB)
	ADC Buffers (1 KB)
	FFT Scratch (2 KB)

ADC Data Acquisition

Ping-Pong Buffer Strategy

```
/* ADC Ping-Pong Buffers */
int16_t gADCSamplesPing[ADC_SAMPLE_SIZE]; // 512 samples
int16_t gADCSamplesPong[ADC_SAMPLE_SIZE]; // 512 samples

volatile bool gPing = true;      // Current buffer indicator
volatile bool gCheckADC = false; // New data ready flag
```

DMA fills one buffer while CPU processes the other, enabling continuous sampling.

Log 300 Data Scaling

```
/* Scales ADC samples and converts them to Q15 format */
void adc_q15_scale(int16_t *samples)
{
    int16_t shift = 2048;  // Mid-scale for 12-bit ADC
    for (int i = 0; i < ADC_SAMPLE_SIZE; ++i)
    {
        // Convert from unsigned 12-bit to signed Q15
        // Shift of 4 provides 16x gain for better SNR
        samples[i] = samples[i]*8
    }
}
```

Feature Extraction

```
#include "feature_extract.h"

// Initialize feature extraction (call once at startup)
FE_init();

// Process each frame of ADC samples
FE_process(newSamples, gNewFeatures, 0);  // varIndex=0 for single channel
```

NPU Inference

```
/* NPU Initialization */
DL_NPU_reset(NPU);
DL_NPU_enablePower(NPU);
while (!(DL_SYSCTL_getStatus() & DL_SYSCTL_STATUS_NPU_READY));

/* Input/Output tensor structures */
struct tvmgen_default_inputs tvm_if_map = { (void*) &gIF_Map[0] };
struct tvmgen_default_outputs tvm_of_map = { (void*) &gOF_Map[0] };

/* Run inference */
tvmgen_default_run(&tvm_if_map, &tvm_of_map);

/* Wait for completion */
while (!tvmgen_default_finished) { __WFE(); }
```

```
/* Output: gOF_Map[0][0] = arc confidence, gOF_Map[0][1] = normal confidence */
```

Trip Logic (Voting Algorithm)

```
#define INFERENCE_FRAME_THRESHOLD_FOR_ARC 5

/* Circular buffer for predictions */
volatile uint8_t gPredVector[8] = {0};
volatile uint8_t gPredPtr = 0;
volatile uint16_t gPredSum = 0;

int trip_logic(int inf)
{
    /* Update circular buffer */
    gPredSum -= gPredVector[gPredPtr];

    if (inf >= 0) {
        gPredVector[gPredPtr] = 1;
        gPredSum += 1;
    } else {
        gPredVector[gPredPtr] = 0;
    }

    gPredPtr = (gPredPtr + 1) % 8;

    /* Majority voting: 5 out of 8 frames must detect arc */
    return (gPredSum > INFERENCE_FRAME_THRESHOLD_FOR_ARC) ? 1 : 0;
}
```

8 Code Examples

Examples Available

Table 8-1.

Example	Location	Description
Arc Fault Detection (NPU)	`examples/ac_arc_fault_detection_ai_npu/`	Complete inference application
Data Capture	`examples/ac_arc_fault_data_capture/`	Training data collection
Live Preview	`examples/ac_arc_fault_detection_live_preview/`	Real-time visualization

Project File Structure

```
ac_arc_fault_detection_ai_npu/LP_MSPM0G5187/
├── ac_arc_fault_detection_ai_npu.c      # Main application
├── ac_arc_fault_detection_ai_npu.syscfg # SysConfig settings
├── ticlang/mspm0g5187.cmd               # Linker script
└── model/
    ├── model.a                           # Compiled NN model
    ├── tvmgen_default.h                  # Model interface
    └── user_input_config.h               # Feature config
```

Figure 8-1. Project File Structure

9 Performance Benchmarks

Note

The performance benchmarks given below are for the ArcFault_model_1400_t with an input configuration of [1,1,256,1].

Table 9-1. Performance Benchmarks

Metric	Value	Notes
Model Size- ArcFault (Flash)	Approximately 5564B	Compiled model.a
Application Code	Approximately 13KB	Including libraries
Tensor Arena (SRAM)	Approximately 2KB	Runtime workspace
Total SRAM Used	Approximately 9KB	Out of 32 KB available
Timing		
ADC Sampling	4.75ms	512 samples at 107kSps
Feature Extraction	~5ms	FFT + binning
NPU Inference	~0.7ms	Model dependent
Total Frame Time	~7.19 ms	Per inference cycle
End-to-End Latency	<150 ms	Including 8-frame voting
Power		
Active Power (MCU)	~8 mW	@ 80 MHz, 3.3 V
Total System	~55 mW	Including TIDA-010971 AFE
Accuracy		
Detection Accuracy	>99%	Typical trained model
False Positive Rate	0.01%	Percentage of negatives wrongly flagged
Precision	99.97%	Percentage of predicted positives correct.
Recall	99.72%	Percentage of actual positives caught
F1-Score	99.84%	Balance of precision and recall.

10 Power Optimization

The reference design uses continuous monitoring for lowest latency, preferred for safety-critical AFCI applications:

- ADC runs continuously at 107kSps
- NPU inference approximately every 9.3ms
- Always-on operation

For battery-powered applications, duty cycling can reduce power by approximately 50% but increases detection latency proportionally. This is not recommended for UL 1699 compliance.

11 Testing and Validation

Required Equipment

Table 11-1. Required Equipment

Equipment	Purpose
Oscilloscope	Waveform capture (≥100MHz)
Current Probe	Current measurement (DC-50MHz)
USB Isolator	PC protection (2500V isolation)

Safety Warning

WARNING
ARC GENERATION SAFETY: Arc generation involves EXTREME HAZARDS including electrocution, severe burns, fire, and toxic fumes.

Only qualified electrical engineers with appropriate PPE should perform arc testing. Work in fire-safe environment with safety observer present. Use current-limited AC source.

Basic Test Procedure

- Power-On: Verify 3.3V rail, confirm MCU boots (LED indication). Note: when using Launchpad, do not power the board from other inputs.
- Normal Operation: Apply various loads, verify no false trips (5 min each)
- Arc Detection: Generate controlled arc, verify detection within 100ms
- Masking Loads: Test dimmers, vacuums, drills - verify NO trip

12 Troubleshooting

Build Errors

Table 12-1. Build Errors

Error	Design
`cannot find -lmodel`	Verify model.a in model/ directory, check linker path

Runtime Issues

Table 12-2. Runtime Issues

Symptom	Design
Same output regardless of input	Verify normalization params match training, check Q15 conversion

Detection Issues

Table 12-3. Detection Issues

Issue	Design
Missed arc detections	Lower voting threshold (4/8), verify sensor bandwidth

Note
Only MSPM0G5187 has an integrated NPU. Other MSPM0 variants cannot use NPU-accelerated inference.

13 Summary

This application note presents a complete reference design for implementing AC arc fault detection using Edge AI on the low-cost MSPM0G5187 microcontroller with integrated Neural Processing Unit (NPU). The design addresses a critical safety challenge: arc faults cause over 30,000 home fires annually, yet traditional circuit breakers and fuses cannot detect them because series arcs may draw less current than normal loads and operate at frequencies (kHz to MHz) that thermal protection devices do not monitor.

The proposed design leverages machine learning inference to overcome the limitations of fixed analog circuits. By combining hardware-accelerated neural network processing with sophisticated signal conditioning from the TIDA-010971 analog front end, the system achieves >99% detection accuracy, <10ms response time, and >99% specificity to masking loads in compliance with UL 1699 safety standards. The architecture employs a four-family CNN model set, with the recommended ArcFault_model_1400_t delivering 99.88% accuracy while consuming only 5.6KB of Flash and 2.4KB of SRAM.

The firmware implementation uses continuous ADC sampling at 107 kSps with ping-pong DMA transfers for uninterrupted data acquisition. Real-time feature extraction via 512-point FFT produces 256 time-frequency features per inference cycle, processed by the NPU in approximately 0.7ms. A majority-voting algorithm (5 of 8 frames) ensures robust detection while minimizing nuisance trips. Total end-to-end latency remains below 150ms, and active power consumption is approximately 8mW for the MCU and 55mW for the complete system. The reference design includes comprehensive guidance on model development, from data collection and labeling strategies through quantization, training, and deployment via the TI Neural Network Compiler. Both Frame Mode and Window Mode labeling approaches are supported to accommodate different datasets and front-end architectures. The provided examples, firmware code, and performance benchmarks enable engineers to develop production AFCI products for residential and commercial electrical systems with minimal development effort while maintaining compliance with National Electrical Code (NEC) requirements and UL 1699 standards. While this reference design provides a strong foundation for AFCI development, production devices require formal UL certification testing and validation. The modular architecture and scalable model options support deployment across a range of performance and cost requirements, from ultra-low-footprint implementations to high-accuracy configurations for complex electrical environments.

14 References

Hardware

1. Texas Instruments, *AC Arc Fault Detection With Edge AI for Circuit Breakers Reference Design*, design guide.
2. Texas Instruments, *MSPM0G5187 Evaluation Module*, EVM user's guide.
3. Texas Instruments, *MSPM0G5187 Mixed-Signal Microcontrollers With TinyEngineTM NPU*, datasheet.

Software

1. Texas Instruments, MSPM0-SDK, software.
2. Texas Instruments, Edge AI Studio, software.
3. Texas Instruments, *TI Neural Network Compiler for MCUs User's Guide -2.1.1 LTS*, webpage.
4. Texas Instruments, Texas Instruments/tinyml-tensorlab, webpage.

Standards

1. UL 1699, Arc-Fault Circuit-Interrupters, standards.
2. NEC Article 210.12, AFCI Protection Requirements, standards.
3. IEC 62606, General Requirements for AFDDs, standards.

15 Terminology

Table 15-1. Terminology

Term	Definition
AFCI	Arc-Fault Circuit Interrupter
DMA	Direct Memory Access
Edge AI	Machine learning inference on edge devices
FFT	Fast Fourier Transform
NPU	Neural Processing Unit
Q15	Fixed-point format with 15 fractional bits
QAT	Quantize-Aware Training
TI-NNC	TI Neural Network Compiler

16 Frequently Asked Questions

Q: Can I use this design for a certified AFCI product?

A: This is a reference design. Production AFCIs require formal UL 1699 certification testing.

Q: Does this work with other MSPM0 devices?

A: NPU-accelerated inference requires MSPM0G5187 (only MSPM0 with integrated NPU). Other variants can run software-only inference without hardware acceleration.

Q: How much training data do I need?

A: Minimum 100 captures per class. Production models typically use 500+ captures per class.

Q: Can I use a different current sensor?

A: Yes. TMCS1123 Hall-effect sensor or current transformers work with appropriate signal conditioning. Ensure DC-MHz bandwidth for arc detection.

Q: Why are inference results always the same?

A: Check normalization parameters in user_input_config.h match training. Verify Q15 conversion and that model.a matches the architecture.

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you fully indemnify TI and its representatives against any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale, TI’s General Quality Guidelines, or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products. Unless TI explicitly designates a product as custom or customer-specified, TI products are standard, catalog, general purpose devices.

TI objects to and rejects any additional or different terms you may propose.

Last updated 10/2025