Getting started with PSOC™ Control C3M/P8, C3M/P7 MCUs on ModusToolbox™ software

About this document

Scope and purpose

This application note introduces the PSC3M/P8, PSC3M/P7 devices, a multi-core, high-performance, low-power, and secure MCU designed for complex power conversion and motor control applications. It helps you explore the PSC3M/P8, PSC3M/P7 devices architecture, the associated development tools, and guides you through creating your first project using the Eclipse IDE for ModusToolbox™ software. This document also highlights additional learning resources available for accelerating your understanding of the PSC3M/P8, PSC3M/P7 family.

Additionally, this application note outlines the device's capabilities, provides an overview of the development ecosystem, and walks you through a simple "Hello World" application to help you become familiar with using the PSC3M/P8, PSC3M/P7 devices. It also demonstrates how to create an application starting from an empty starter template, with the completed design provided as a code example. For hardware design considerations, see AN242230 - Hardware design guidelines for PSOC™ Control C3M/P8, C3M/P7 MCUs.

Intended audience

This document is intended for those who are new to PSC3M/P8, PSC3M/P7 devices and ModusToolbox™ software.

Associated part family

All PSC3M/P8, PSC3M/P7 devices.

Software version

ModusToolbox™ software 3.9 or above.

Table of contents

About this document . . . 1
Table of contents . . . 2
1 Introduction . . . 4
1.1 Architecture and product lines . . . 4
1.2 PSC3M/P8, PSC3M/P7 devices features . . . 6
1.3 Target applications . . . 12
2 PSC3M/P8, PSC3M/P7 resources . . . 15
3 PSC3M/P8, PSC3M/P7 devices development kits . . . 16
4 PSC3M/P8, PSC3M/P7 MCUs ecosystem for firmware/application development . . . 17
4.1 Installing the ModusToolbox™ tools package . . . 17
4.2 Choosing an IDE . . . 17
4.3 ModusToolbox™ help . . . 17
5 Getting started with PSC3M/P8, PSC3M/P7 MCUs design . . . 19
5.1 Prerequisites . . . 19
5.1.1 Hardware . . . 19
5.1.2 Software . . . 19
5.2 PSC3M/P8, PSC3M/P7 MCUs multicore application structure . . . 20
5.3 About the design . . . 21
5.4 Application development instructions . . . 21
5.4.1 Eclipse IDE for a ModusToolbox™ . . . 22
5.4.1.1 Create a new application . . . 22
5.4.1.2 View and modify the design . . . 26
5.4.1.2.1 Open the Device Configurator and configure the system . . . 29
5.4.1.2.2 Add retarget-io middleware . . . 32
5.4.1.3 Write firmware . . . 33
5.4.1.4 Build the application . . . 44
5.4.1.5 Program the device . . . 44
5.4.1.6 Test your design . . . 47
5.4.1.7 Debugging the application using KitProg3/MiniProg4 . . . 49
5.4.2 Visual Studio Code (VS Code) for ModusToolbox™ . . . 51
5.4.3 IAR Embedded Workbench for ModusToolbox™ user guide . . . 51
5.4.4 Keil µVision for ModusToolbox™ user guide . . . 51
5.5 Working with PSC3M/P8, PSC3M/P7 MCU devices main CPU and PPCA CPUs . . . 52
5.5.1 Initialization of resources . . . 52
5.5.2 Communication between CPUs . . . 52
6 Summary . . . 53
References . . . 54

Table of contents

Glossary 55

Revision history 56

Trademarks 57

Disclaimer 58

1 Introduction

The PSC3M/P8, PSC3M/P7 microcontrollers mainly targeted for implementation of complex power conversion solutions, such as totem-pole power factor correction (PFC), half-bridge LLC, full-bridge LLC, phase-shift full-bridge as well as their interleaved versions, and many other topologies that use wide bandgap (WBG) devices and motor control solutions involving multiple motors in various configurations.

The PSOC™ Control C3 MCU integrates the following features on a single chip:

- 180 MHz 32-bit Arm® Cortex® -M33 CPU with
  - Floating-point unit (FPU)
  - Digital signal processor (DSP)
  - Memory protection unit (MPU)
  - TrustZone® framework support
  - 16 KB instruction cache (I-cache)
- Programmable power control accelerator (PPCA) subsystem with
  - Two 200 MHz 32-bit Arm® Cortex® -M33 CPUs
  - 112 KB SRAM with ECC support
  - Four 12-bit, 25 Msps SAR ADC modules, 24 direct channels plus 16 multiplexed channels
  - Nine differential comparator slope generators and fast analog comparators with <10 ns built-in 12-bit DAC supporting a refresh rate of maximum 50 MHz
  - Two 11-bit buffered R2R DAC supporting a refresh rate of maximum 2 MHz
  - Two coordinate rotation digital computer (CORDIC) blocks
  - 12 x 32-bit high-resolution PWMs (HRPWM), <100 ps, with complementary outputs
  - 14 x 16-bit PWM running at CPU clock resolution with complementary output
  - Configurable and programmable events processing unit (EPU) block for handling signals
  - Two independent instances of 3P3Z hardware filters with total 6 channels
  - Custom logic block unit (CLB) implementing digital current observer and matrix converter
- Up to 512 KB read-while-write flash with ECC support
- 64 KB ROM for boot code
- 128 KB SRAM with ECC support
- 2 x 32-bit and 2 x 16-bit pulse-width modulator (TCPWM)
- Seven serial communication blocks (SCB) supporting UART, I2C, and SPI protocols, two with extended PMBus® features and two CAN FD (one supporting 8 Mbps)
- Low-power operation: Sleep, Deep Sleep (three modes) below 12 µA, and Hibernate below 1 µA
- Up to 60 GPIOs with programmable drive modes, strengths, and slew rates, and 26 analog pins
- Security: Platform Security Architecture (PSA) L3 certified; configurable flash partitioning and protection
- Class B and SIL 2 compliant safety test libraries are available
- ModusToolbox™ development environment with installable SDKs and libraries, industry standard Arm® tools, and RTOS support

1.1 Architecture and product lines

Figure 1 shows a detailed block diagram of the MCU.

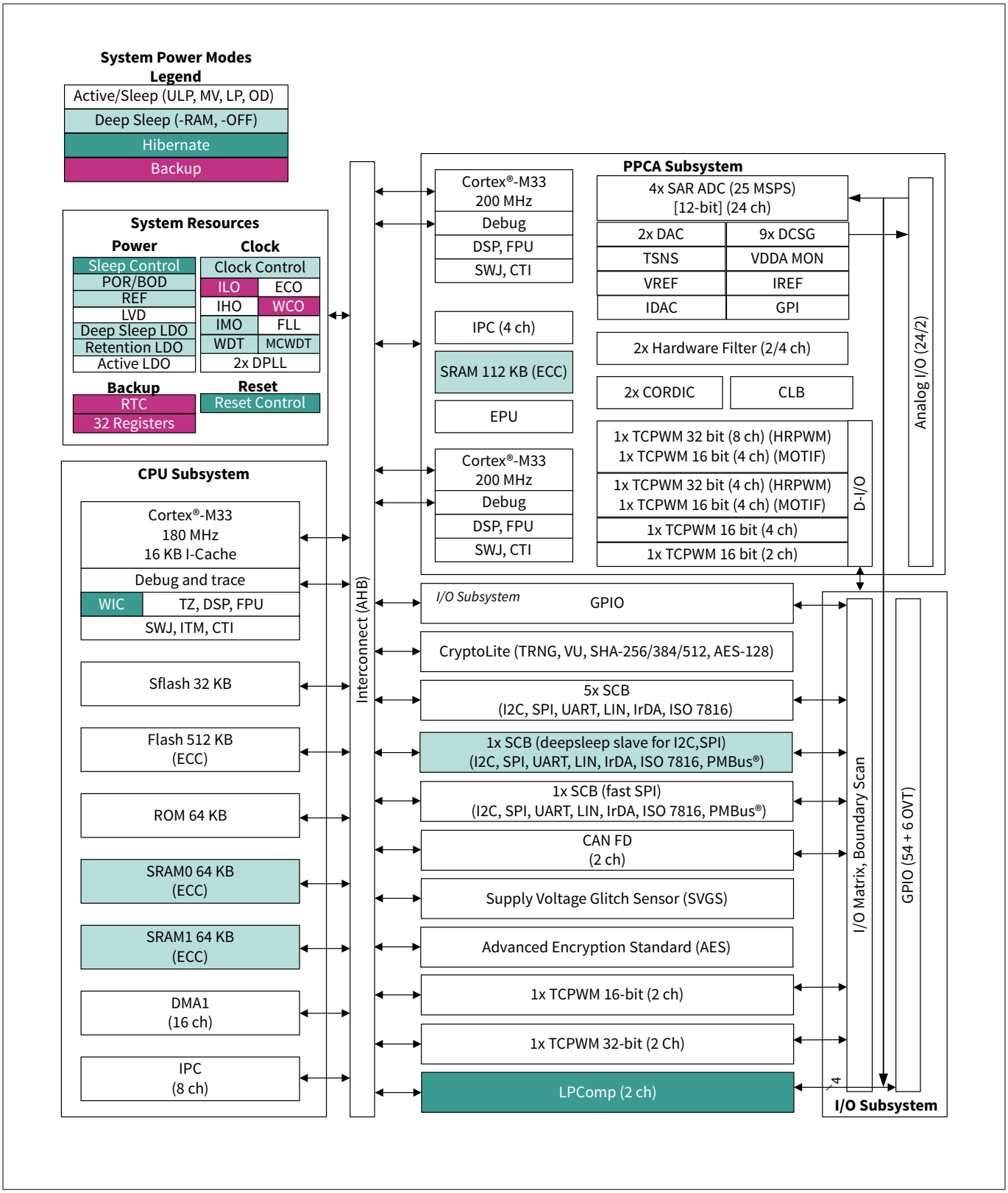


Figure 1 Functional block diagram

There are four variants in the PSC3M/P8, PSC3M/P7 devices. Table 1 provides an overview of different components available in the product line.

Table 1 Product variants

PSC3P8x	PSC3M8x	PSC3P7x	PSC3M7x
• 180 MHz Arm® Cortex®-M33 main core, up to 512 KB flash and 256 KB RAM • Dual core 200 MHz PPCA Arm® Cortex®-M33, 25 Msps ADC, HRPWM	• 180 MHz Arm® Cortex®-M33 main core, up to 512 KB flash and 256 KB RAM • Dual core 200 MHz PPCA Arm® Cortex®-M33, 25 Msps ADC, HRPWM, CORDIC, MOTIF	• 180 MHz Arm® Cortex®-M33 main core, up to 512 KB flash and 256 KB RAM • Single core 200 MHz PPCA Arm® Cortex®-M33, 25 Msps ADC, HRPWM	• 180 MHz Arm® Cortex®-M33 main core, up to 512 KB flash and 256 KB RAM • Single core 200 MHz PPCA Arm® Cortex®-M33, 25 Msps ADC, HRPWM, CORDIC, MOTIF

1.2 PSC3M/P8, PSC3M/P7 devices features

This device supports the following features:

- **CPU subsystem**:
  - Arm® Cortex®-M33 processor running at up to 180 MHz
  - Digital signal processor (DSP), floating-point unit (FPU), memory protection unit (MPU), and 16 KB instruction cache (I-cache)
  - One direct memory access (DMA) controller with 16 channels
- **Security**:
  - Some devices are Platform Security Architecture Level 2 (PSA L2) certified, and some are PSA L3 certified
  - Features secured boot, secured firmware update, secured debug, and secured RMA mode for field failure
  - Hardware and software isolation with TrustZone®
  - Infineon proprietary protection with protection context, memory protection controller (MPC) and peripheral protection controller (PPC)
  - Cryptography accelerator
    - Hardware acceleration for symmetric (AES-128, AES-256)
    - Vector unit (VU) to accelerate asymmetric key cryptography (ex: RSA and ECC)
    - Hash functions support SHA-256,SHA-384, SHA-512
    - True random number generator (TRNG) function
    - ECC Signature verification: ECDSA NIST P-256, ECDSA NIST P-384, ECDSA NIST P-521
  - Supply voltage glitch sensor (SVGS) designed to detect supply-voltage manipulations by attackers attempting to disrupt device operation
  - DICE ready: Device identifier composition engine (DICE) mechanism support
- **Memory**:
  - On-chip flash memory:
    - Up to 512 KB flash with error correction code (ECC) support, RWW capability, 64 KB ROM for boot code, and bootloader functions, 32 KB reserved for the supervisory region
    - Built-in device firmware upgrade (DFU) support in the boot ROM via a serial interface (UART)
  - Static random access memory (SRAM):
    - 128 KB SRAM with ECC support

- Organized as two independent 64 KB blocks (SRAM0 and SRAM1)
- Full 64 KB + 64 KB SRAM is available in Deep Sleep mode
- Protected SRAM data path using hardware ECC for soft error detection and correction

- **Clocking subsystem**:
  - 8 MHz IMO with Deep Sleep operation, offering ±2% accuracy
  - 48 MHz internal high-frequency oscillator (IHO) with ±1% accuracy
  - 4 to 35 MHz external crystal oscillator (ECO) with a phase-locked loop (PLL) for CPU and subsystem clocking
  - 32 kHz external watch crystal oscillator (WCO), usable for real-time clock (RTC) applications
  - 32 kHz internal low-frequency oscillator (ILO) with ±10% accuracy
  - External clock (EXTCLK): Maximum frequency 100 MHz
  - One frequency lock loop (FLL) with 25 MHz to 100 MHz output range
  - Two digital phase-locked loops, DPLL#0 and DPLL#1, with 50 MHz to 200 MHz output range
- **Low-power operation**:
  - Operates at low power with a voltage range of 1.71 V to 3.63 V
  - Supports six power modes for fine-grained power management:
    - Active
    - Sleep
    - Deep Sleep
    - Deep Sleep-RAM
    - Deep Sleep-OFF
    - Hibernate
  - Four power profiles (Ultra Low Power, Medium Frequency, Low Power, Overdrive) to optimize speed vs power consumption in Active mode
- **Communication peripherals**:
  - **Serial communication blocks (SCBs)**:
    - Up to five independent run-time-reconfigurable SCBs:
      - Each SCB is software-configurable as I^2C, SPI, or UART in master or slave mode
    - Two additional run-time-reconfigurable SCBs with PMBus® protocol support:
      - Software-configurable as master or slave for I^2C, SPI, or UART protocols
      - PMBus® protocol features:
        - Baud rates of up to 1 Mbps
        - Timeout event generation (TGS) for wrong or stuck commands
        - Hardware address screening with support for a minimum of five slave address masks (MSAs) to reject or NACK incoming address commands
        - Compliance with PMBus® and SMBus specifications
      - One SCB instance supports wakeup from Deep Sleep mode with PMBus® extensions
    - **Additional features**:
      - One SCB supports operation in Deep Sleep mode and wakeup from Deep Sleep in I^2C slave and SPI slave modes

    - One SCB supports fast SPI operation with speeds of up to 50 MHz
    - SCBs support single-wire, half-duplex mode for UART communication
  - **CAN FD**:
    - Supports up to two CAN FD channels
    - Each channel operates at speeds of up to 8 Mbps
- **I/O subsystem**:
  - **Programmable GPIO pins**:
    - Supports up to 86 functional pins, including 60 digital GPIOs, 24 dedicated analog-only inputs (14 of the 60 GPIOs are analog inputs capable), and two dedicated analog pins configurable as inputs or outputs only
    - Provides programmable drive modes, strengths, and slew rates
    - Six of the 60 GPIOs as *GPIO_OVT* to support PMBus®/SMBUS compliance with input leakage specifications
    - Up to four *GPIO_OVT* pins can be routed to analog inputs of ATOP and used by the ADC or comparator
- **Low-power comparator (LPComp)**:
  - Supports a pair of low-power comparators
  - Provides comparator thresholds for comparison, which can be sourced either from an external pin or an internal fixed reference of approximately 0.6 V ± 0.2 V
- **Timer/counter pulse-width modulator (TCPWM)**:
  - One group provides two 32-bit TCPWM channels
  - Another group provides two 16-bit TCPWM channels
  - Both TCPWM outputs and inputs are exclusively connected to the event processing unit (EPU) and trigger MUX
  - One instance is clocked using a stable clock source and can be used for monitoring the main CPU
  - The second instance is clocked using a robust clock source to ensure reliable heart-beat monitoring of the PPCA and can be used for monitoring the PPCA CPUs
- **Programmable power control accelerator subsystem (PPCA)**:
  - PPCA CPU subsystem:
    - **CPUs**:
      - Two Arm® Cortex®-M33 processors (CPU0 and CPU1), each operating at up to 200 MHz
      - Digital signal processor (DSP), floating-point unit (FPU)
    - **CPUs memories**:
      - Five instances of SRAM memory modules are available
      - SRAM supports ECC, allowing them to detect two errors and correct one error
      - Four local memory blocks with the following sizes: 32 KB, 16 KB, 32 KB, and 16 KB. The first two blocks are assigned to the first PPCA CPU (CPU0) for code and data. The other two blocks are allocated to the second PPCA CPU (CPU1) for code and data. If CPU1 is not used, these blocks serve as code extension and data extension for CPU
      - A 16 KB shared memory block is accessible by both PPCA CPU0 and PPCA CPU1. This memory can also be used as a data extension for either CPU0 or CPU1

- **IPC block**:
  - An IPC block facilitates communication between PPCA CPU0 and PPCA CPU1. This block operates through the shared 16 KB SRAM
  - The IPC block supports one group, four channels, and four interrupts
- **AMBA bus infrastructure**:
  - Includes a bus matrix, five AHB slave inputs, and 12 AHB master outputs

- **Analog TOP subsystem (ATOP SS)**:
  - **Analog-to-digital converter (ADC)**:
    - Four 12-bit (greater than 10-bit ENOB) SAR ADCs (ADC0 - ADC3) with a conversion rate of up to 25 Msps
    - Supports up to 12 dedicated differential analog input channels or up to 24 dedicated single-ended input channels
    - An option to connect 16 multiplexed additional analog inputs to two analog input channels
    - Input voltage range: –0.4 V to 3.3 V
    - Enables simultaneous sampling, arbitrary sampling, or continuous programmable sampling sequences
    - A selectable internal or external 1.2 V ADC voltage reference
    - An analog front end (AFE) with selectable input gain values of ×8.25, ×16.5, ×33, and ×66
    - Initiates the start of conversion (SOC) via a dedicated hardware signal per channel, per group of channels, or by writing into the ADC trigger register
    - Aligns the end of sampling (EOS) with the provided SOC or by using the control ADC conversion FSM
    - Provides a configurable ADC trigger source from the event processing unit (EPU)
  - **Digital comparators (DCMP)**:
    - Features digital comparator outputs directed to the EPU
    - Up to 12 digital comparators with configurable digital thresholds
    - Generates an event when the input signal crosses the programmed threshold
    - Supports configurable comparison methods including immediate comparison, integration of immediate comparison, accumulation of immediate comparisons, and averaged comparisons
  - **ADC filters (ADC FILTs)**:
    - The ADC embeds a configurable set of digital filters, including a median filter, linear interpolation/extrapolation filter, low-pass filter, cascaded integrator third-order comb filter, configurable average filter, detection of minimum values in the input data stream, and detection of maximum values in the input data stream
    - Supports up to 64× oversampling, providing 15-bit effective resolution on up to six ADC channels
    - Filters can be chained together and optionally sourced from the output of another ADC core
    - The MIN/MAX module can be sourced either by the input or the output of the median filter
  - **Differential analog comparator with slope generator (DCSG)**:
    - Features 9 active, configurable differential or single-ended comparators with a worst-case comparator delay of 5 ns
    - A differential analog comparator with an embedded DAC for programmable thresholds or references
    - Employs a dynamic latch comparator architecture, supporting a maximum clock frequency of 200 MHz

  - Three comparators with selectable digital slope generator
  - Supports a selectable input voltage range of 0 V to 3.3 V or ±200 mV
  - Ensures a maximum jitter of one clock cycle and a propagation delay of less than two clock cycles
  - Offset matches DAC resolution, providing ±0.7 mV for a 3.3 V rail-to-rail range and a 12-bit DAC
  - Supports 12-bit DAC threshold settings with differential non-linearity (DNL) less than four LSB
  - Ensures 10-bit monotonicity and a maximum DAC refresh rate of 50 Msps (minimum 20 Msps)
  - Guarantees less than 10 clock cycles settling time for 400 code jumps with an accuracy of ±8 LSB
  - Integrates a DAC refresh trigger function with a blanking time of less than 12 clock cycles
  - Supports A versus B comparisons
  - Provides comparator trigger inputs and outputs connected to the event processing unit (EPU) module
- **Digital-to-analog converter (DAC) R2R**:
  - Two 12-bit single-ended DACs with a refresh rate of 2 MHz
  - Ensures 10-bit monotonicity
  - Selectable DAC output ranges, including 0 V to VREF (1.2 V), 0 V to VDDA/2, ±0.2 V, or ±VDDA
  - Provides a selectable buffered DAC output full-scale range of 0.1 V to VDDA/2 or 0.1 V to VREF
  - Features a 50 pF load driver
- **Additional features**:
  - Temperature sensor (TSNS) with an accuracy of ±5°C
  - Provides a current reference (AREF) for ATOP subsystem resources
  - Voltage reference (VREF) for ATOP SS resources
  - Features a current digital-to-analog converter (IDAC) module to generate a selectable, known current for sensing external resistance
  - General-Purpose Input (GPI) module to convert an analog input into a digital output by using fixed analog thresholds
- **Event processing unit (EPU)**:
  - Facilitates flexible management of events generated by peripherals to create triggers
  - Supports trigger generation through a linear combination of processed input events and allows routing of input triggers as interrupts to the PPCA CPUs
  - Configurable and programmable event processing system, comprising 32 Type-1 processing units and 16 Type-2 processing units
  - Processes selected inputs connected to events generated by CPUs and peripheral resources, with outputs connected to triggers of CPUs and peripheral resources
  - Supports various functionalities, including asynchronous operation, synchronous bypass, input polarity inversion, and rising-edge extraction
  - Integrates filtering capabilities such as masking filters, blanking filters, de-bouncing filters (through integration), and pulse delays of up to 128 clock cycles
- **Hardware filter (HWFILT3P3Z)**:
  - Supports two instances of the hardware filter (HWFILT3P3Z), each of which can be allocated to one of the PPCA CPUs
  - The first instance supports four 3Poles3Zeros (3P3Z) hardware filters
  - The second instance supports two 3Poles3Zeros (3P3Z) hardware filters
  - Features a 16-bit input and a 24-bit output

- A 40-bit multiplier and a 44-bit internal accumulator
- Ensures execution latency of up to eight CPU clock cycles
- Provides support for arithmetic 2's complement and saturation operators
- Programmable dedicated data path for enhanced performance

- **Custom logic block (CLB)**:
  - The CLB module provides dedicated hardware support for specific functions or control algorithms that require high computational resources
  - Examples of supported functions include digital observers used in Totem Pole PFC controllers and matrix converter commutation
  - Additionally, the CLB module supports trajectory control and spread-pulse start-up functionalities
- **Coordinate Rotation Digital Computer (CORDIC)**:
  - Supports all CORDIC operating modes for solving circular (trigonometric) and hyperbolic functions
  - Integrates independent lookup tables to accelerate calculations for enhanced performance
- **Timer/counter pulse-width modulation (TCPWM)**:
  - Four instances of TCPWM (TCPWM0 - TCPWM3) are available within the PPCA
  - All counters (PERIOD, DUTY, CC0, CC1, DT) support greater than or equal to (≥) match as well as equal to (=) match conditions
  - Center-aligned, edge modes
  - Shadow update of duty, period, dead-time, output signal polarity
  - Comparator-based triggering of kill signals
  - Trigger inputs for all instances are connected to the event processing unit (EPU) outputs, while trigger outputs for all instances are connected to the EPU inputs
  - All four instances operate synchronously with the PPCA CPUs and run on the PPCA clock, with a maximum frequency of 200 MHz
  - HRPWM feature for period, duty, and dead-time insertion with a typical resolution of less than 100 ps
  - **First TCPWM instance (TCPWM0)**:
    - Eight 32-bit counters with high-resolution PWM (HRPWM) functionality
    - Parallel input data path for CC0 and CC1 registers enabled for four 32-bit counters
    - Four 16-bit counters, with the 16-bit counter group supporting MOTIF
    - PWM outputs for all 16-bit and 32-bit counters are connected to HSIOM
  - **Second TCPWM instance (TCPWM1)**:
    - Four 32-bit counters with HRPWM functionality
    - Parallel data path for CC0 and CC1 enabled for two 32-bit counters
    - Four 16-bit counters, with the 16-bit counter group supporting MOTIF
  - **Third TCPWM instance (TCPWM2)**:
    - Four 16-bit counters
    - Trigger inputs and outputs are connected to the EPU
    - Trigger outputs for PWM outputs are also connected to the trigger MUX
  - **Fourth TCPWM instance (TCPWM3)**:
    - Two 16-bit counters
    - Trigger inputs and outputs are connected to the EPU
    - Trigger outputs for PWM outputs are also connected to the trigger MUX

1.3 Target applications

The versatile, secured, low-power, feature-rich offerings in the PSC3M/P8, PSC3M/P7 devices make it an ideal choice for a wide variety of end applications. Some of these applications are listed below.

- High power and high performances SMPS for telecom and server
- Multiphase high-power SMPS for hyperscale
- Power tools
- Home appliances
- Industrial drives
- Light electric vehicles
- LED lighting
- Solar inverters, and much more

The ModusToolbox™ software environment supports PSC3M/P8, PSC3M/P7 MCUs application development with a set of tools for configuring the device, setting up peripherals, and complementing your projects with world-class middleware. See the Infineon GitHub repos for board support packages (BSPs) for all kits, libraries for popular functionality like motor control and power stage converter, and a comprehensive array of example applications to get you started.

PSC3M/P8, PSC3M/P7 devices are highly capable and flexible solution for industrial applications. For example, the presence of the PPCA components like CLB, HRPWM, ATOP, Hardware 3P3Z filter, etc. makes implementation of complex power conversion solutions possible.

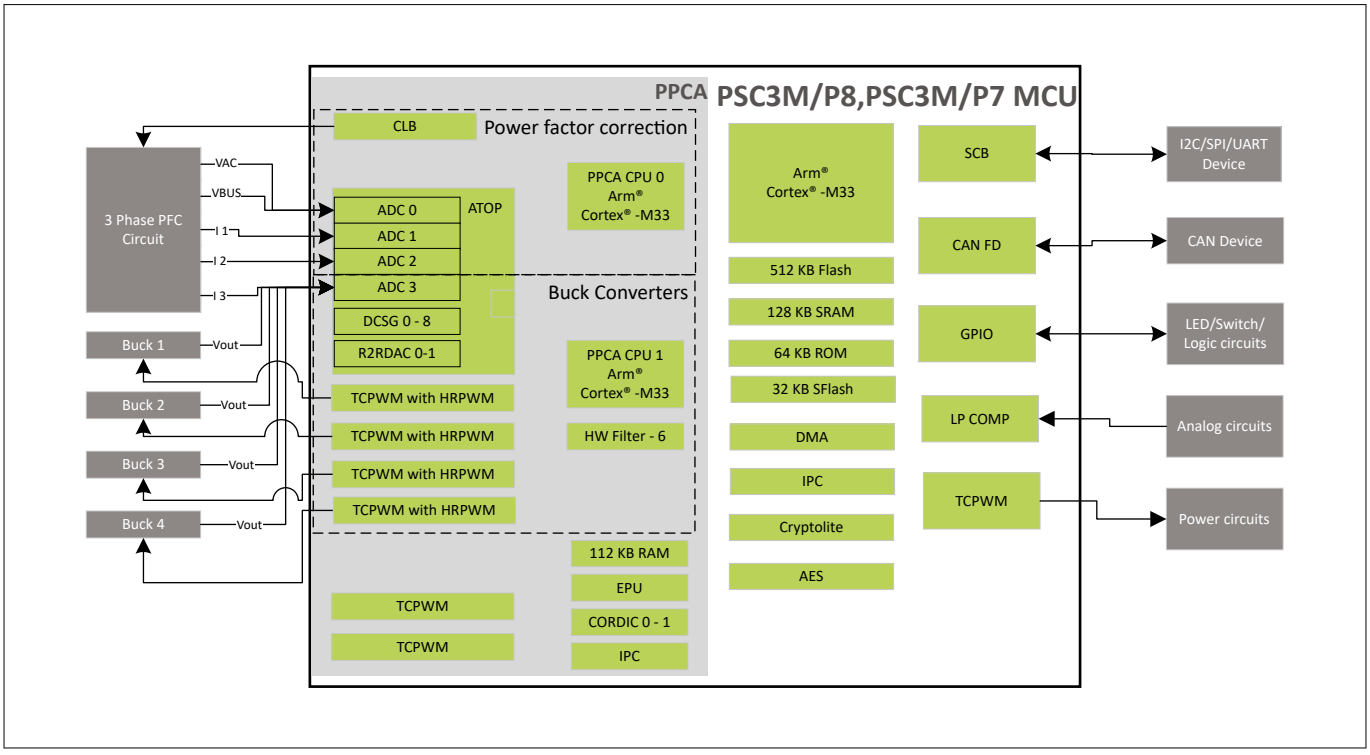


Figure 2 Application-level block diagram using PSC3M/P8, PSC3M/P7 devices for power conversion

Figure 2 shows a power conversion solution by taking advantage of the following features:

- Custom logic block (CLB) in PPCA makes it possible to implement PFC with minimum CPU usage
- High-speed ADC in PPCA ATOP subsystem helps PFC implementation
- Events processing unit (EPU) in PPCA helps to route trigger signals between CLB and ATOP
- Additional ADC channels, HRPWM, and dedicated data path in the PPCA helps buck converters

- LPComp to compare the analog signals for protection
- Inbuilt security features for firmware protection
- Serial communication blocks (SCBs) to interface with external devices like motion sensors
- External TCPWM blocks to interface additional power driver circuits
- CAN FD channels to communicate with CAN FD capable peripherals

Presence of the TCPWM blocks with MOTIF feature and high-speed ADCs helps implementation of complex motor control solutions using PSC3M/P8, PSC3M/P7 devices. Figure 3 shows another use case that makes use of the following features to implement multiple motor control solutions with ease.

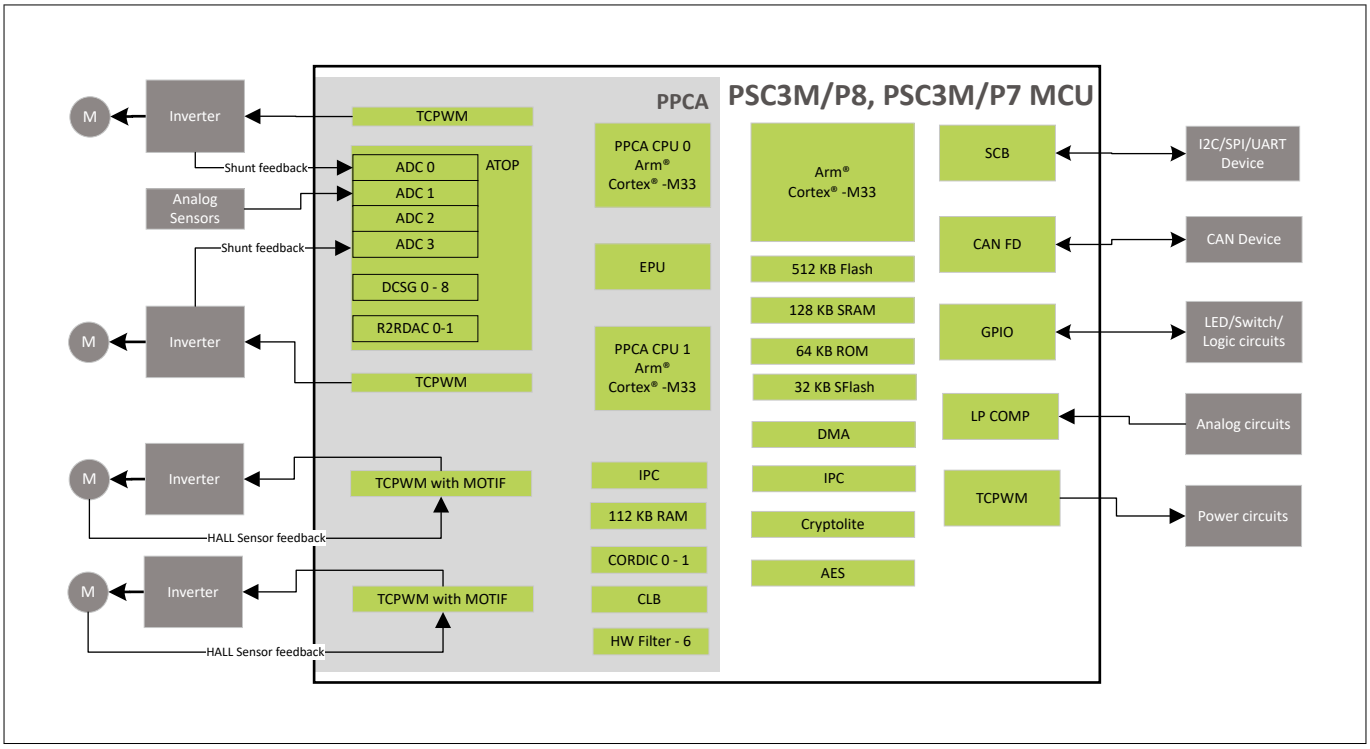


Figure 3 Application-level block diagram using PSC3M/P8, PSC3M/P7 devices for motor control

- Two MOTIF instances available with PPCA TCPWM help to run two motors without CPU intervention
- Availability of high speed ADCs along with other resources helps control of additional motors
- Presence of additional ADC channels and DCSG in ATOP helps implementation of efficient protection mechanism
- Events processing unit (EPU) in PPCA helps to route trigger signals between TCPWM and ATOP
- LPComp to compare the analog signals for protection
- Inbuilt security features for firmware protection
- Serial communication blocks (SCBs) to interface with external devices like motion sensors
- External TCPWM blocks to interface additional power driver circuits
- CAN FD channels to communicate with CAN FD capable peripherals

In addition to the features shown in the above two use cases, the presence of the following component in PSC3M/P8, PSC3M/P7 devices make it suitable for implementation of much more complex solutions.

- IPC for communication between the CPUs
- Cryptolite and AES for data protection
- DMA for data exchange
- PPCA CORDIC for fast motor control trigonometric operations

- PPCA hardware Filter for implementation of efficient digital filters
- ATOP R2R DAC for generating analog signals

2 PSC3M/P8, PSC3M/P7 resources

A wealth of technical resources is available to develop applications with the PSC3M/P8, PSC3M/P7. These resources are listed below.

- Overview: PSC3M/P8, PSC3M/P7 webpage
- Product selectors: PSOC™ Control C3 Performance Line
- Performance line datasheet describes the device family and provides electrical specifications
- Application notes and code examples cover a broad range of topics, from basic to advanced level. You can also browse our collection of code examples
- Reference manuals (architecture and register) provide detailed descriptions of the architecture and registers in each device family
- Development tools: Many low-cost kits are available for evaluation, design, and development of different applications using PSC3M/P8, PSC3M/P7 devices
- Training videos: Video training on our products and tools, including PSC3M/P8, PSC3M/P7 devices
- Technical support: PSOC™ Control C3 community forum

3 PSC3M/P8, PSC3M/P7 devices development kits

Infineon provides a wide variety of hardware development kits in various form factors and features to enable easy and rapid evaluation and prototyping of PSC3M/P8, PSC3M/P7 devices-based applications.

The following table lists the PSC3M/P8, PSC3M/P7 kits with various features.

ModusToolbox™ software is the software development platform for creating embedded applications using the development kits.

Table 2 Development kits

Kit MPN	Product	Kit name	BSP
KIT_PSC3M8_EVK	PSC3M8x	PSOC™ Control C3M8 Evaluation Kit	BSP
KIT_PSC3M8_CC1	PSC3M8x	PSOC™ Control C3M8 Digital Power Control Card	BSP

For the complete list of kits for the PSC3M/P8, PSC3M/P7 MCUs, see the PSC3M/P8, PSC3M/P7 webpage.

4 PSC3M/P8, PSC3M/P7 MCUs ecosystem for firmware/ application development

Infineon provides the ModusToolbox™ software for firmware/application development on MCUs. ModusToolbox™ software is a modern, extensible development ecosystem supporting a wide range of Infineon microcontroller devices, including PSOC™ Arm® Cortex® MCUs, TRAVEO™ T2G Arm® Cortex® MCUs, XMC™ Industrial MCUs, AIROC™ Wi-Fi devices, AIROC™ Bluetooth® devices, and USB-C Power Delivery MCUs. This software includes configuration tools, low-level drivers, middleware libraries, and other packages that enable you to create MCU and wireless applications. All tools run on Windows, macOS, and Linux. ModusToolbox™ includes an Eclipse IDE, which provides an integrated flow with all the ModusToolbox™ tools. Other IDEs such as Visual Studio Code, IAR Embedded Workbench, and Arm® MDK (µVision) are also supported.

ModusToolbox™ software supports stand-alone device and middleware configurators. Use the configurators to set the configuration of different blocks in the device and generate code that can be used in firmware development.

Libraries and enablement software are available at the GitHub site.

ModusToolbox™ tools and resources can also be used on the command-line tool. See the build system chapter in the ModusToolbox™ tools package user guide for detailed documentation.

4.1 Installing the ModusToolbox™ tools package

Refer to the ModusToolbox™ tools package installation guide for details.

4.2 Choosing an IDE

ModusToolbox™ software, the latest-generation toolset, is supported across Windows, Linux, and macOS platforms. ModusToolbox™ software supports 3rd-party IDEs, including the Eclipse IDE, Visual Studio Code, Arm® MDK (µVision), and IAR Embedded Workbench. The tools package includes an implementation for all the supported IDEs. The tools support all PSC3M/P8, PSC3M/P7 devices. The associated BSP and library configurators also work on all three host operating systems.

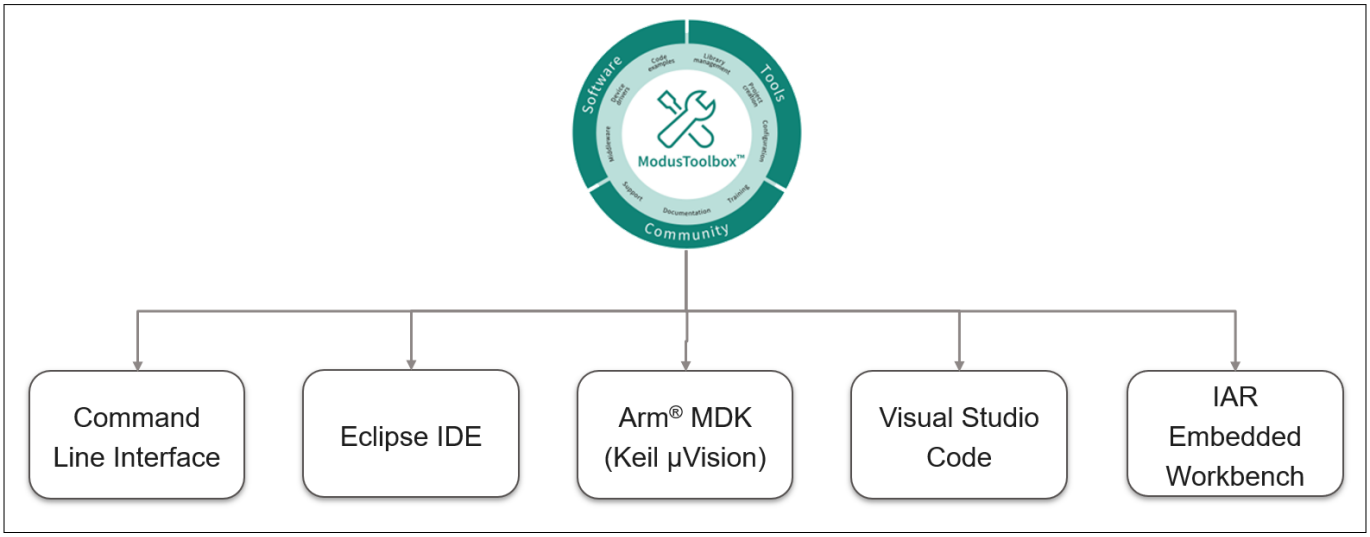


Figure 4 ModusToolbox™ environment

4.3 ModusToolbox™ help

The ModusToolbox™ ecosystem provides documentation and training. Launch the Eclipse IDE for ModusToolbox™ software and navigate to the following Help menu items:

Choose **Help** > **ModusToolbox™ General Documentation**:

- ModusToolbox™ Documentation Index: Provides brief descriptions and links to various types of documentation included as part of the ModusToolbox™ software
- ModusToolbox™ Installation Guide: Provides instructions for installing the ModusToolbox™ software
- ModusToolbox™ User Guide: This guide primarily covers the ModusToolbox™ aspects of building, programming, and debugging applications. Additionally, it covers various aspects of the tools installed along with the IDE
- ModusToolbox™ Training Material: Links to the training material available at https://github.com/Infineon/training-modustoolbox
- Release notes: Describes the features and known limitations for the ModusToolbox™ software, provided as part of the ModusToolbox™ tools package included with the installer

For documentation on Eclipse IDE for ModusToolbox™, choose **Help** > **Eclipse IDE for ModusToolbox™ documentation**.

- User Guide: Provides descriptions about creating applications as well as building, programming, and debugging them using Eclipse IDE
- Eclipse IDE Survival Guide: This is a link to a forum with answers for questions about how to get common tasks done

5 Getting started with PSC3M/P8, PSC3M/P7 MCUs design

This section does the following:

- Demonstrate how to build a simple PSC3M/P8, PSC3M/P7 MCU-based design and program into the development kit
- Demonstrate the implementing functionality using main CPU and PPCA CPUs
- Makes it easy to learn PSC3M/P8, PSC3M/P7 MCUs design techniques and how to use the ModusToolbox™ software with different IDEs

5.1 Prerequisites

Before you get started with the application development instructions, make sure that you have the appropriate development kit for your PSC3M/P8, PSC3M/P7 MCUs product line and have installed the required software. You also need an internet connection to access the GitHub repositories during project creation.

5.1.1 Hardware

The following design example is developed for the PSC3M/P8, PSC3M/P7 Evaluation Kits (KIT_PSC3M8_EVK). However, you can build the application for other development kits also. For more details, see the section Application development instructions.

5.1.2 Software

ModusToolbox™ software 3.9 or above.

After installing the software, see the ModusToolbox™ tools package user guide to get an overview of the software.

5.2 PSC3M/P8, PSC3M/P7 MCUs multicore application structure



Figure 5 PSC3M/P8, PSC3M/P7 MCUs multicore project structure

As shown in the Figure 5, PSC3M/P8, PSC3M/P7 devices multicore project contains three different sub projects. The main CM33 core of the MCU is with TrustZone® security feature, the project is made completely secure

application. CM33 CPUs in the PPCA subsystem are without the security feature, so the other two projects are non-secure projects for the PPCA cores.

According to the design, the project structure can be change to 3/4/1 folder structure. Majority of the released code examples in the GitHub follow 3 (three) folder structure. Few code example with 4 (four) and (1) single folder structure are also available.

All sub projects contain folders called build, deps, and libs and source code(main.c) and Makefile. These are local to the sub projects in where it is located. Settings specific to the sub project can be made in the local Makefile.

The source code that needs to be run on a specific core can be added to the sub project specific to the core. For example, source code added to the sub project main code non-secure project will run on the main core in a non-secure context. Similarly, the secure code that needs to be run on the main core can added to the main core secure project. Code added to the PPCA core 0 non-secure project and PPCA core 1 non-secure project will be run on PPCA Core 0 and PPCA Core 1, respectively.

The parent project contains folders, bsps, build, deps, and templates and files common_app.mk, common.mk, and Makefile. These are applicable for all sub projects. The build folder contains the final build artifact that can be downloaded to the target.

Details of the individual components in the project will be explained in the coming sections.

5.3 About the design

This design uses the PSC3M/P8, PSC3M/P7 devices to execute two tasks: UART communication and LED control.

After device reset, the main CPU in the MCU initializes and starts the PPCA and its CPUs. The main CPU uses the UART to print messages to the serial port stream. Each PPCA core toggles a shared memory variable every second, and the main core monitors these variables to control LEDs and print status messages via UART, creating a synchronized LED blinky pattern across all three cores. LED1 indicate PPCA core 1, LED 2 indicate PPCA core 2, and LED 3 indicate main core activity.

5.4 Application development instructions

Instructions are grouped into several sections. Each section is dedicated to a phase of the application development workflow. The major sections are:

1. Create a new application
2. View and modify the design
3. Write firmware
4. Build the application
5. Program the device
6. Test your design

This design is developed for the PSC3M/P8, PSC3M/P7 Evaluation Kit.

As mentioned the section Choosing an IDE, the ModusToolbox™ software supports the following IDEs:

1. Eclipse IDE
2. Visual Studio Code (VS Code)
3. Keil µVision
4. IAR Embedded Workbench

The following sections provide details on how to create a new application on different IDEs.

5.4.1 Eclipse IDE for a ModusToolbox™

5.4.1.1 Create a new application

This section takes you on a step-by-step guided tour of the new application creation process. It uses the Multicore Empty App starter application and manually adds the functionality from the Multicore Blinky App starter application.

If you are familiar with developing projects with ModusToolbox™ software, you can use the Multicore Blinky App starter application directly. It is a complete design, with all the firmware written for the supported kits. You can walk through the instructions and observe how the steps are implemented in the code example.

If you start from scratch and follow all the instructions in this application note, you can use the Multicore Blinky App code example as a reference while following the instructions.

Launch the Dashboard 3.9 application to get started.

***Note**: Dashboard 3.9 application needs access to the internet to successfully clone the starter application onto your machine.*

The Dashboard 3.9 application helps you get started using the various tools with easy access to documentation and training material, a simple path for creating applications and creating and editing BSPs.

1. Open the Dashboard 3.9 application

   To open the Dashboard 3.9 application, do one of these:

   - Windows: Navigate to `[ModusToolbox installation path]/tools_3.9/dashboard/dashboard.exe` or you can also select the "ModusToolbox™ Dashboard 3.9" item from the Windows Start menu
   - Linux: [ModusToolbox™ installation path]/tools_3.9/dashboard and run the executable
   - macOS: Run the "dashboard" app

2. On the Dashboard 3.9 window, in the right pane, in the Target IDE drop-down list, select Eclipse IDE for ModusToolbox™, and click Launch Eclipse IDE for ModusToolbox™

   ***Note**: When using ModusToolbox™ 3.9, Eclipse IDE needs to be installed separately.*

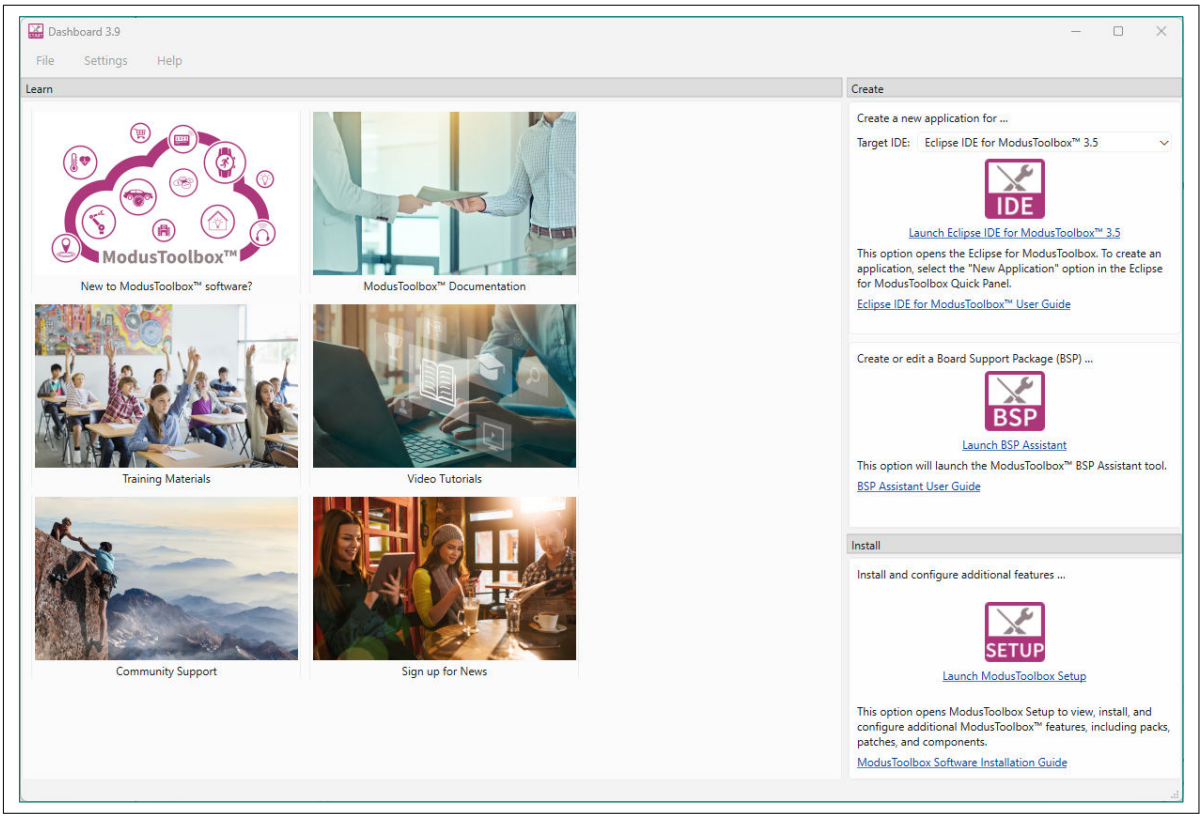


Figure 6 Dashboard 3.9 application

3. Select a new workspace

At launch, Eclipse IDE for ModusToolbox™ displays a dialog to choose a directory for use as the workspace directory. The workspace directory is used to store workspace preferences and development artifacts. You can choose an existing empty directory by clicking the Browse button, as shown in the Figure 7. Alternatively, you can type in a directory name to be used as the workspace directory along with the complete path, and the IDE will create the directory for you

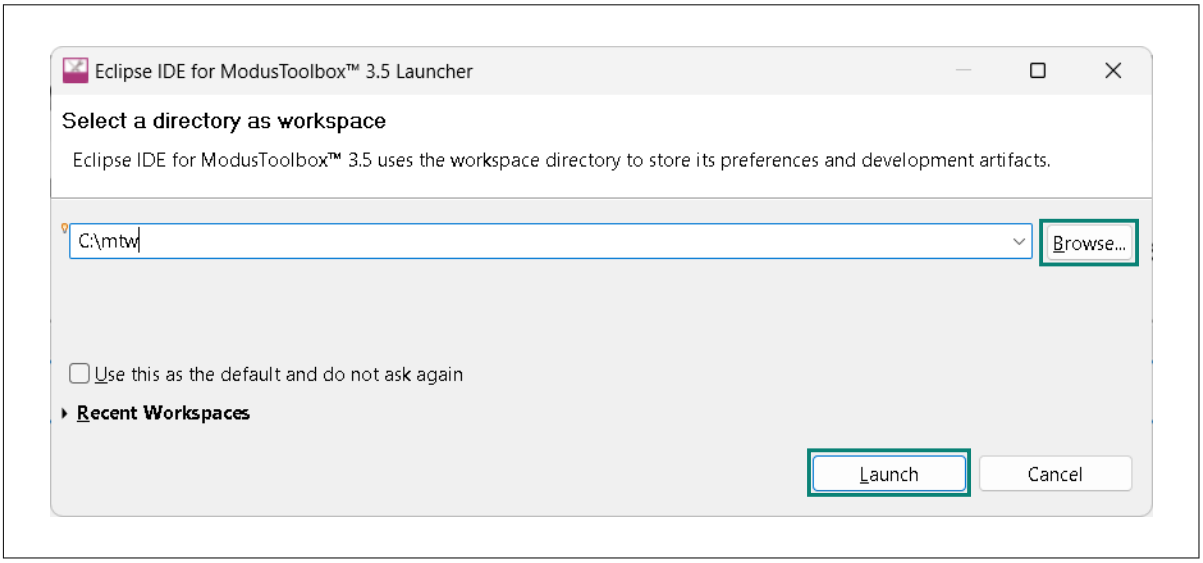


Figure 7 Select a directory as the workspace

4. Create a new ModusToolbox™ application

a. Click New Application in the Start group of the Quick Panel

b. Alternatively, you can choose **File** > **New** > **ModusToolbox™ Application**, as Figure 8 shows. Displays the Eclipse IDE for ModusToolbox™ Application window

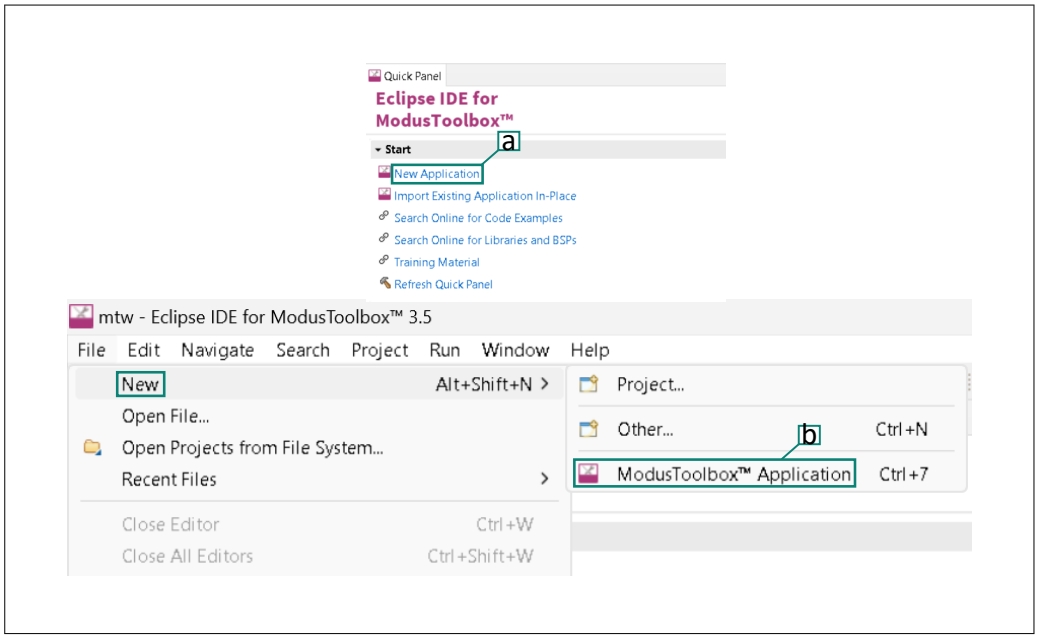


Figure 8 Create a new application

5. Select a target PSOC™ Control C3M8 Evaluation Kit

ModusToolbox™ speeds up the development process by providing BSPs that set various workspace/ project options for the specified development kit in the new application dialog

a. In the Choose Board Support Package (BSP) dialog, choose the Kit Name that you have. The steps that are followed use `KIT_PSC3M8_EVK`, as shown in Figure 9

b. Click Next

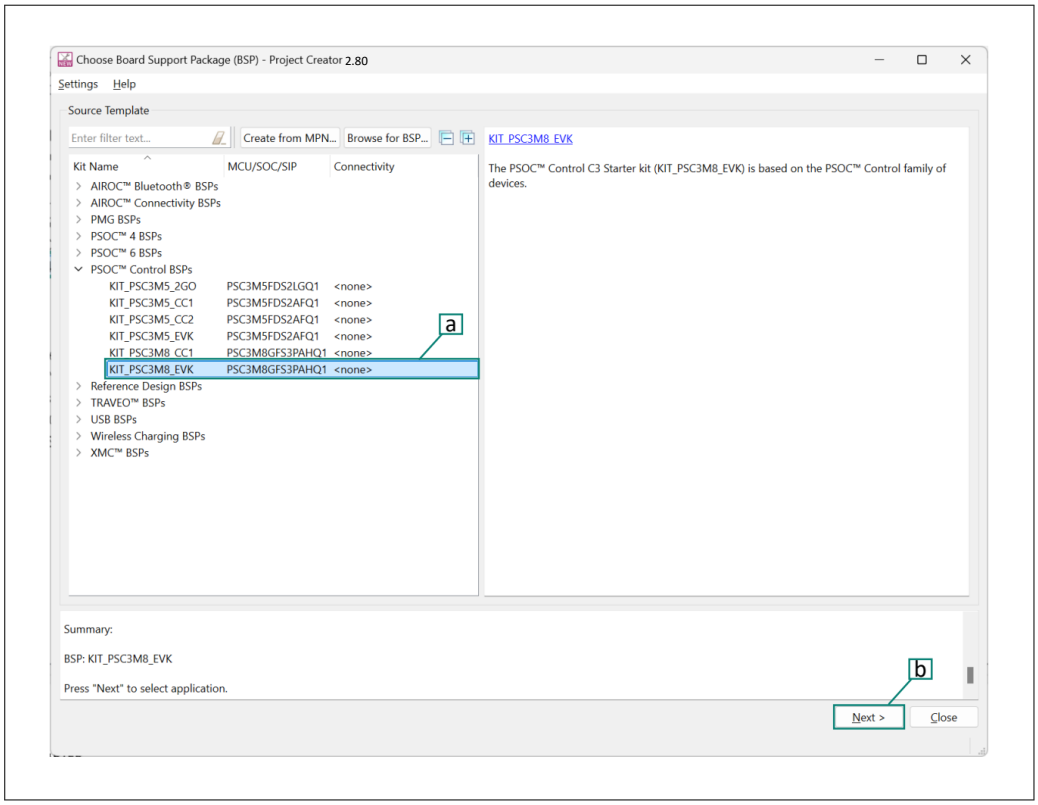


Figure 9 Choose target hardware

c. In the Select Application dialog, select Multicore Empty App starter application, as shown in the Figure 10

d. In the Name field, type in a name for the application, such as `Hello_World`. You can choose to leave the default name if you prefer

***Note**: Try to use a short name without spaces in between*

e. Click Create to create the application, as shown in the Figure 10, wait for the Project Creator to automatically close once the project is successfully created

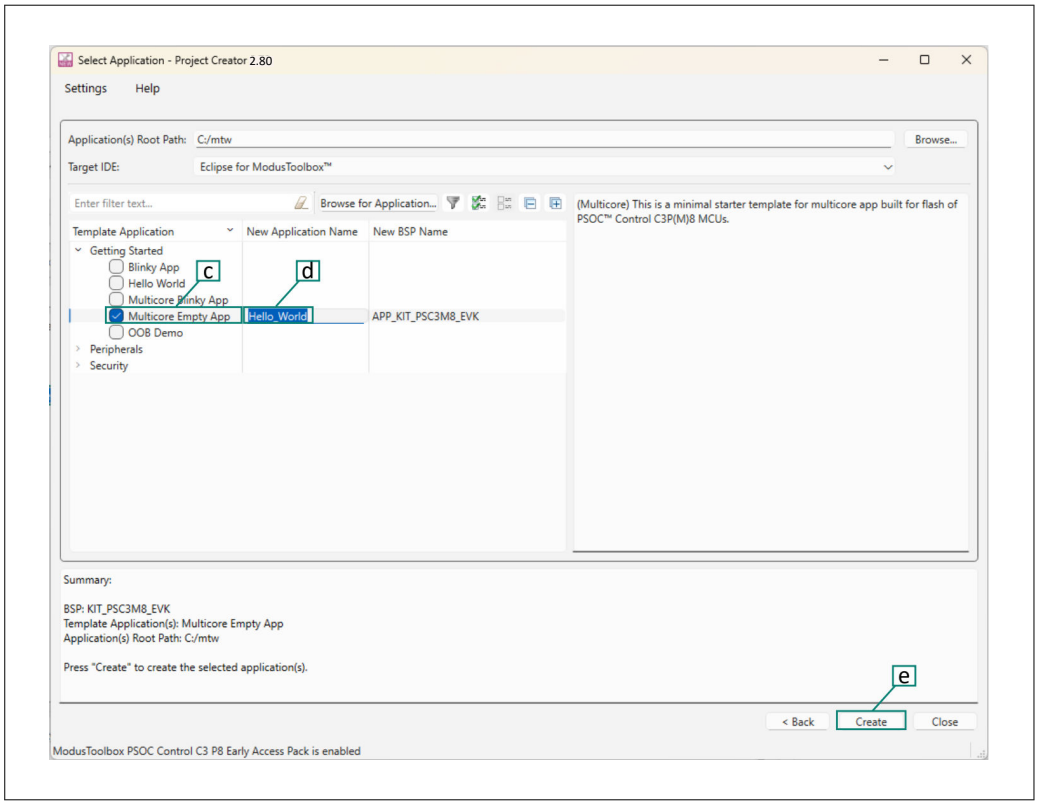


Figure 10 Choose starter application

You have successfully created a new ModusToolbox™ application for a PSC3M/P8, PSC3M/P7 devices.

The BSP uses PSC3M8GFS3PAHQ1 as the default device that is mounted on the PSOC™ Control C3M8 Evaluation Kit.

If you are using custom hardware based on the PSOC™ Control C3M8 or a different PSC3M/P8, PSC3M/P7 part number, see the "Creating your Own BSP" section in the ModusToolbox™ user guide

5.4.1.2 View and modify the design

Figure 11 shows the ModusToolbox™ project explorer interface displaying the structure of the application project.

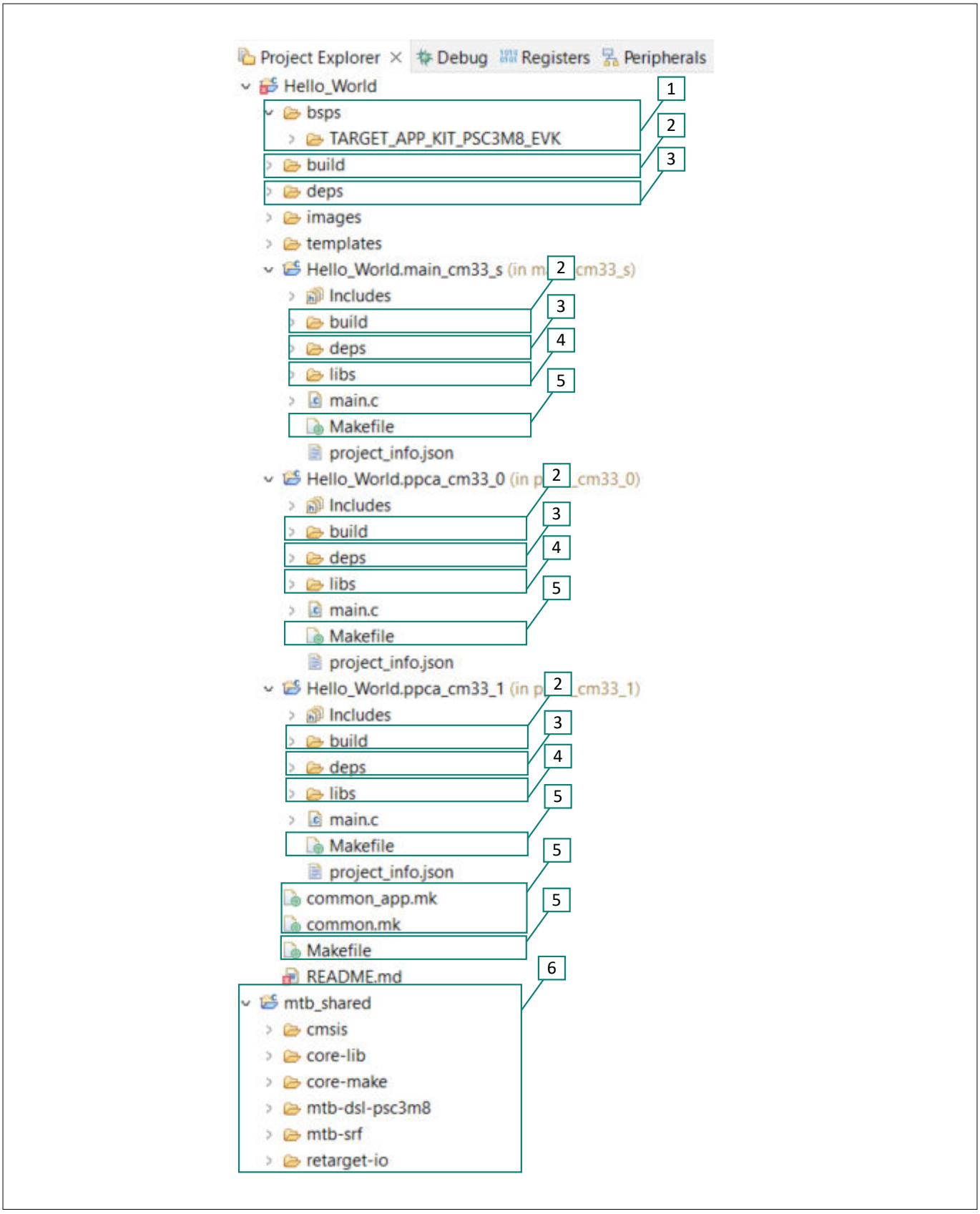


Figure 11 Project Explorer view

A project folder consists of various subfolders – each denoting a specific aspect of the project.

1. The files provided by the BSP are in the ‘bsps’ folder and are listed under TARGET_<bsp name> subfolders. All the input files for the device and peripheral configurators are in the config folder inside the BSP. The Generated Source folder in the BSP contains the files that are generated by the configurators and are prefixed with `cycfg_`. These files contain the design configuration as defined by the BSP. You can directly customize configurator files of BSP for your application rather than overriding the default design configurator files with custom design configurator files since BSPs are completely owned by the application.

   The BSP folder also contains the linker scripts and the start-up code for the PSOC™ Control C3M8 MCU used on the board. You can see that the bsps folder will be located under the main project and it will not be available under the sub projects
2. The ‘build’ folder contains all the artifacts resulting from a build of the project. The output files are organized by target BSPs

   Here we can see the build folder under the main project and the sub projects. The build folder of the main project contains the final executable. build folder inside the subprojects contains the intermediate files from that project
3. The ‘deps’ folder contains .mtb files, which provide the locations from which ModusToolbox™ pulls the libraries that are directly referenced by the application. These files typically each contain the GitHub location of a library. The .mtb files also contain a git Commit Hash or Tag that tells which version of the library is to be fetched and a path as to where the library should be stored locally

   For example, here, retarget-io.mtb points to `mtb://retarget-io#latest-v1.X#$$ASSET_REPO$$/retarget-io/latest-v1.X`. The variable $$ASSET_REPO$$ points to the root of the shared location which defaults to `mtb_shared`. If the library must be local to the application instead of shared, use $$LOCAL$$ instead of `$$ASSET_REPO$$`

   Like the build folder, we can see deps folder also in the main project and the subprojects. Here the deps folder contains the dependencies for the application where it resides
4. The libs folder also contains .mtb files. In this case, they point to libraries that are included indirectly as a dependency of a BSP or another library. For each indirect dependency, the Library Manager places a .mtb file in this folder. These files have been populated based on the targets available in deps folder

   For example, using BSP KIT_PSC3M8_EVK populates the libs folder with the following .mtb files: cmsis.mtb, core-lib.mtb, core-make.mtb, deivce-db.mtb, mtb-dsl-psc3m8.mtb

   The libs folder contains the file mtb.mk, which stores the relative paths of all the libraries required by the application. The build system uses this file to find all the libraries required by the application

   Everything in the libs folder is generated by the Library Manager so you should not manually edit anything in that folder
5. An application contains a Makefile which is at the application's root folder. This file contains the set of directives that the make tool uses to compile and link the application project. There can be more than one project in an application. In that case there is a Makefile at the application level and one inside each project

   As we are developing application for a multi-core MCU and one core in itself with both secure and non secure modes, we have sub projects in the application. Here. we can see Makefiles for all subprojects and the parent project. The setting specific to the sub project can be made inside the respective Makefile. Setting that needs to be applied to the entire app resides under the main project
6. By default, when creating a new application or adding a library to an existing application and specifying it as shared, all libraries are placed in an `mtb_shared` directory adjacent to the application directories

   The `mtb_shared` folder is shared between different applications within a workspace. Different applications may use different versions of shared libraries if necessary

5.4.1.2.1 Open the Device Configurator and configure the system

BSP configurator files are in the `bsps/TARGET_<BSP-name>/config` folder. Click <Application-name> from Project Explorer then click the Device Configurator link in the Quick Panel to open the file design.modus in the Device Configurator as shown in the Figure 12. You can also open other configuration files in their respective configurators or click the corresponding links in the Quick Panel.

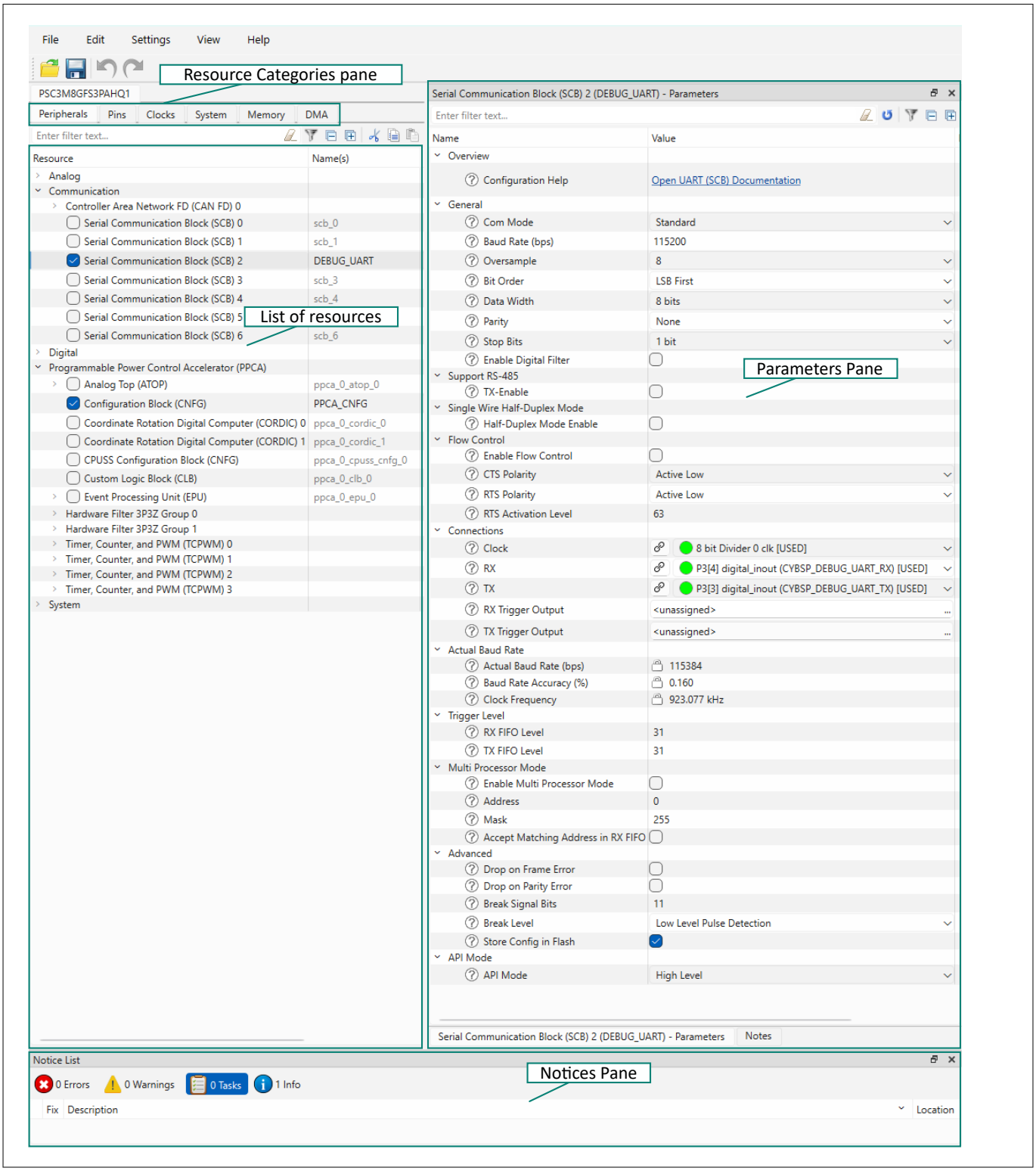


Figure 12 Device Configurator-UART configuration

The Device Configurator provides a set of Resources Categories tabs. Here you can choose between different resources available in the device such as peripherals, pins, and clocks from the List of Resources.

You can choose how a resource behaves by choosing a Personality for the resource. For example, a serial communication block (SCB) resource can have EZI2C, I2C, SPI, or UART personalities. The Name(s) is your name for the resource, which is used in firmware development. One or more aliases can be specified by using a comma to separate them (with no spaces).

The Parameters pane is where you enter the configuration parameters for each enabled resource and the selected personality. The Code Preview pane shows the configuration code generated per the configuration parameters selected. This code is populated in the cycfg_ files in the Generated Source folder. The Parameters pane and Code Preview pane may be displayed as tabs instead of separate windows but the contents will be the same.

Any errors, warnings, and information messages arising out of the configuration are displayed in the Notices pane. Configuring the peripheral is required for both PDL and HAL based implementations to work.

1. UART Configuration

   Figure 12 also shows that the SCB 2 is enabled and configured. As the SCB 2 is used for communicating with the user through the debug UART terminal, replicate the same configuration in your setup for the Hello World application

2. PPCA resource allocation

   As PSC3M/P8, PSC3M/P7 devices has a main CPU core and two PPCA CPU cores, configure the system to make the PPCA peripheral available to access by the PPCA CPU cores. In this design PPCA Cores are enabled and started. These two cores toggles a shared memory variable in regular intervals, this data is being used by the main core to print it on the UART console and control LEDs. In this case, all the PPCA resources are allocated to Core 0, see Figure 13

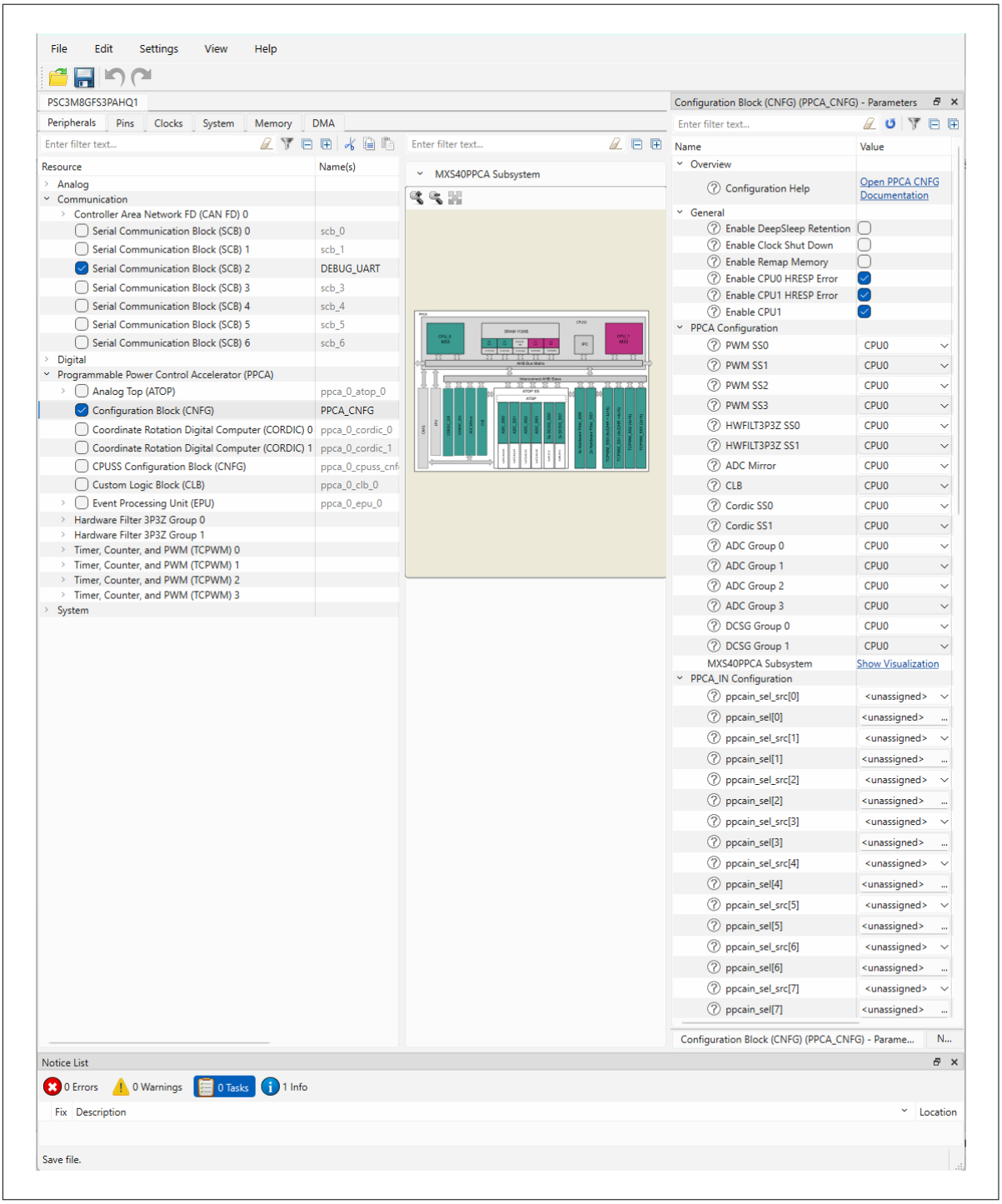


Figure 13 PPCA resource configuration

3. GPIO Configuration

Figure 14 shows that a GPIOs connected to an LEDs on the EVK is configured in output mode to toggle it. For that, enable the pin P1.5(LED 1), P1.7(LED 3), P4.1(LED 2) in the Pins tab in the resource categories pane. Also, make sure that the Drive Mode is set to ‘Strong Drive. Input buffer off’, and Secure attribute to "secure access (0)".

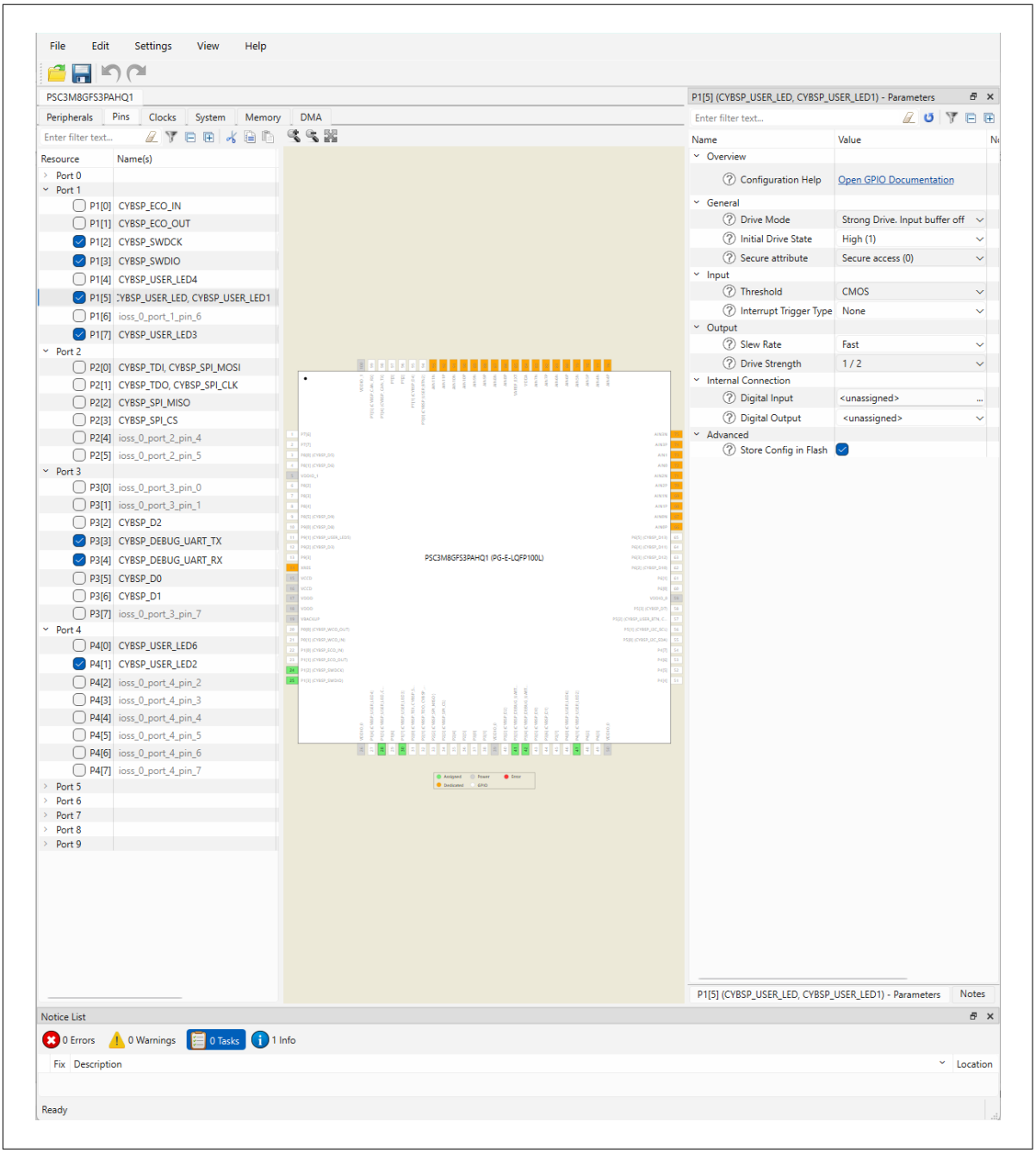


Figure 14 GPIO configuration

At this point in the development process, the required middleware is ready to be added to the design. The only middleware required for the Hello World application is the retarget-io library.

5.4.1.2.2 Add retarget-io middleware

In this section, you will add the retarget-io middleware to redirect standard input and output streams to the UART configured by the BSP. The initialization of the middleware will be done in the main.c file.

1. In the Quick Panel, click the Library Manager link
2. In the subsequent dialog, click Add Libraries
3. Under Peripherals, select and enable retarget-io
4. Click OK and then Update

The files necessary to use the retarget-io middleware are added in the `mtb_shared > retarget_io folder`, and the .mtb file is added to the deps folder, as shown in the Figure 15.

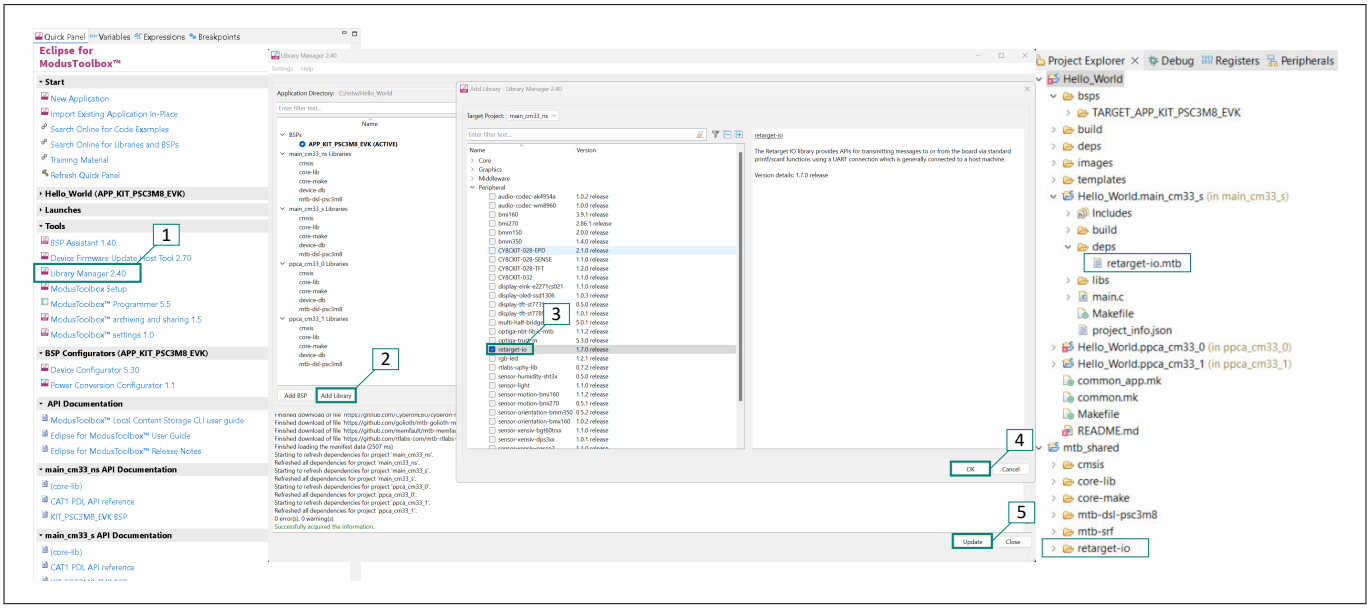


Figure 15 Add the retarget-io middleware

5.4.1.3 Write firmware

At this point in the development process, you have created an application with the assistance of an application template, configured the peripherals using the device configurator, and modified it to add the retarget-io middleware. In this section, you will write the firmware that implements the design functionality.

If you are working from scratch using the Multicore Empty App for PSOC™ Control C3M8 application, you can copy the code snippet provided in this section to the main.c files of the respective sub projects of the application. If you are using the Multicore Blinky App code example, all the required files are already in the application.

Write firmware

Examine the code in the main.c files of the application. Figure 16 shows the firmware flowchart.

After reset, resource initialization for this example is performed by the Main CM33 CPU. It configures the system clocks, pins, clock to peripheral connections, and other platform resources. Then the clocks and system resources are initialized by the BSP initialization function. The retarget-io middleware is configured to use the debug UART, and the user LED is initialized. The debug UART prints a "Multicore Blinky LED Pattern" message on the terminal emulator – the onboard KitProg3 acts as the USB-to-UART bridge to create the virtual COM port.

After the message, the main core CPU initializes a counter variable and two pointers to the shared memory to communicate with the PPCA CPU cores. Then initialize and start PPCA CPU cores 0 and 1. The PPCA CPU cores also initialize pointers to the shared memory and start toggling data in the infinite loop every second.

The main CPU on the same time enters to the infinite loop, prints the data in the shared memory (updated by the PPCA CPUs). Based on the shared memory variable main core toggles an LED1 for PPCA core 0 and LED2 for PPCA core 1 and LED 3 for the main core on the EVK. The PDL delay function is used for providing the time delay for that.

***Note**: This application code uses BSP/HAL/middleware/PDL functions to execute the intended functionality.*

`cybsp_init()` – This BSP function initializes all the system resources of the device, including but not limited to the system clocks and power regulators.

`Cy_SCB_UART_Init()` – This function initializes the debug UART.

`Cy_SCB_UART_Enable()` - This function enables the debug UART.

`mtb_hal_uart_setup() and cy_retarget_io_init()` – These functions set up the HAL UART and redirect the input/output stream to the debug UART.

`Cy_System_Init_CPU0() and Cy_System_Init_CPU1()` - Initialize and start PPCA CPUs.

`Cy_GPIO_Inv()` - Invert the state of a GPIO pin. This is used for toggling the LED.

`Cy_SysLib_Delay()` - Providing a delay between the periodic prints and LED state change.

Copy the following code snippet to the main.c file of your application project.

main.c file for main CPU

```
/*******************************************************************************
 * Header Files
 ******************************************************************************/

#include "cy_pdl.h"
#include "cybsp.h"
#include "cy_retarget_io.h"

/*******************************************************************************
* Macros
*******************************************************************************/

/* Flash memory addresses where PPCA core images are stored */
#define CORE0_IMAGE_ADDRESS   CYMEM_CM33_0_S_m33s_ppca0_nvm_C_S_START
#define CORE1_IMAGE_ADDRESS   CYMEM_CM33_0_S_m33s_ppca1_nvm_C_S_START

#define WORD_OFFSET           0x04
#define IMAGE_SIZE            (0x8000 - WORD_OFFSET)

/* Shared memory addresses for inter-core communication */
/* PPCA cores write to these locations, main core reads from them */
/* Variables located in M4 shared memory space (16KB at 0x20040000-0x20043FFF from PPCA view) */
/* Main core accesses PPCA memory through PPCA peripheral base with memory windows: */
/* M1 (CPU0 data): 0x53020000, M3 (CPU1 data): 0x53040000, M4 (shared): 0x53050000 */
#define PPCA_CPU0_M4_VAR_ADDRESS   0x53050400  /* Written by PPCA Core 0 */
#define PPCA_CPU1_M4_VAR_ADDRESS   0x53050800  /* Written by PPCA Core 1 */

/* Timing configuration */
#define POLLING_DELAY_MS      10     /* Main loop polling interval */
#define LED3_PERIOD_MS        500    /* LED3 heartbeat period */
#define LED3_COUNTER_MAX      (LED3_PERIOD_MS / POLLING_DELAY_MS)

/*******************************************************************************
* Global Variables
*******************************************************************************/

/* Debug UART variables */
static cy_stc_scb_uart_context_t    DEBUG_UART_context; /* DEBUG_UART context */
static mtb_hal_uart_t               DEBUG_UART_hal_obj; /* Debug DEBUG_UART HAL object */

/*******************************************************************************
* Function Prototype
*******************************************************************************/


/*******************************************************************************
* Function Name: main
********************************************************************************
* Summary:
* Main CM33 Core application:
*    1. Boots PPCA Core 0 and Core 1
```

```
*    2. Monitors shared memory variables from both PPCA cores
*    3. Controls LEDs based on PPCA core activity:
*       - LED1: Toggles when Core 0 changes its variable (1 sec period)
*       - LED2: Toggles when Core 1 changes its variable (1 sec, 500ms offset)
*       - LED3: Heartbeat showing main core is alive (500ms period)
*    4. Prints activity messages to UART terminal
*
* Parameters:
*  void
*
* Return:
*  int
*
*******************************************************************************/

int main(void)
{
    cy_rslt_t result;

    /* Initialize the device and board peripherals */
    result = cybsp_init();

    /* Board init failed. Stop program execution */
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }

    /* Initialize UART hardware for debug output */
    Cy_SCB_UART_Init(DEBUG_UART_HW, &DEBUG_UART_config, &DEBUG_UART_context);
    Cy_SCB_UART_Enable(DEBUG_UART_HW);

    /* Setup the HAL DEBUG_UART */
    result = mtb_hal_uart_setup(&DEBUG_UART_hal_obj, &DEBUG_UART_hal_config,
                                &DEBUG_UART_context, NULL);

    /* HAL DEBUG_UART init failed. Stop program execution */
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }

    /* Initialize redirecting of low level IO (printf) to UART */
    result = cy_retarget_io_init(&DEBUG_UART_hal_obj);

    /* retarget IO init failed. Stop program execution */
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }

    /* Transmit header to the terminal */
    /* \x1b[2J\x1b[;H - ANSI ESC sequence for clear screen */
```

```
    printf("\x1b[2J\x1b[;H");
    printf("*************************************************************\r\n");
    printf("PSOC Control C3M8: Main Core and PPCA Cores App\r\n");
    printf("*************************************************************\r\n\n");

    /* enable interrupts */
    __enable_irq();

    // Initialize and enable PPCA configuration
    Cy_PPCA_CNFG_Init(PPCA_CNFG_HW, &PPCA_CNFG_config);
    Cy_PPCA_Enable(PPCA_CNFG_HW);

    /* Boot PPCA Core 0 and Core 1 from flash memory */
    Cy_System_Init_CPU0((void*)CORE0_IMAGE_ADDRESS, IMAGE_SIZE);
    Cy_System_Init_CPU1((void*)CORE1_IMAGE_ADDRESS, IMAGE_SIZE);

    /* Display multicore application banner */
    printf("PPCA CM33 Cores Boot Complete!\r\n\r\n");
    printf("========================================\r\n");
    printf("  Multicore Blinky LED Pattern\r\n");
    printf("========================================\r\n");
    printf("  LED1: PPCA Core0 (1 sec toggle)\r\n");
    printf("  LED2: PPCA Core1 (1 sec toggle, staggered)\r\n");
    printf("  LED3: Main Core  (500ms heartbeat)\r\n");
    printf("========================================\r\n\r\n");

    /* Pointers to shared memory locations for inter-core communication */
    uint32_t *ppca_core0_var = (uint32_t *)PPCA_CPU0_M4_VAR_ADDRESS;
    uint32_t *ppca_core1_var = (uint32_t *)PPCA_CPU1_M4_VAR_ADDRESS;

    /* Counter for LED3 heartbeat timing (500ms period with 10ms polling) */
    uint32_t led3_counter = 0;

    /* Previous values to detect changes in shared variables */
    uint32_t prev_core0_val = *ppca_core0_var;
    uint32_t prev_core1_val = *ppca_core1_var;

    printf("Monitoring PPCA cores activity...\r\n");

    for (;;)
    {
        /* Monitor PPCA Core 0 activity */
        /* Check if Core 0 has changed its shared variable */
        if(*ppca_core0_var != prev_core0_val)
        {

            /* Toggle LED1 to indicate Core 0 activity (EVK kit only) */
            Cy_GPIO_Inv(CYBSP_USER_LED1_PORT, CYBSP_USER_LED1_PIN);
            /* Print status update with LED state */
            printf("\r\n[PPCA Core0] Value: %ld | LED1: %s",
                    (long)*ppca_core0_var,
                    (const char*)((*ppca_core0_var == 1) ? "ON " : "OFF"));
```

```
            /* Update previous value to detect next change */
            prev_core0_val = *ppca_core0_var;
        }

        /* Monitor PPCA Core 1 activity */
        /* Check if Core 1 has changed its shared variable */
        if(*ppca_core1_var != prev_core1_val)
        {
            /* Toggle LED2 to indicate Core 1 activity (EVK kit only) */
            Cy_GPIO_Inv(CYBSP_USER_LED2_PORT, CYBSP_USER_LED2_PIN);
            /* Print status update with LED state */
            printf("\r\n[PPCA Core1] Value: %ld | LED2: %s",
                    (long)*ppca_core1_var,
                    (const char*)((*ppca_core1_var == 1) ? "ON " : "OFF"));

            /* Update previous value to detect next change */
            prev_core1_val = *ppca_core1_var;
        }

        /* Main core heartbeat - LED3 toggles every 500ms */
        /* Counter increments every 10ms, so LED3_COUNTER_MAX counts = 500ms */
        if(led3_counter >= LED3_COUNTER_MAX)
        {
            Cy_GPIO_Inv(CYBSP_USER_LED3_PORT, CYBSP_USER_LED3_PIN);
            led3_counter = 0;
        }
        led3_counter++;

        /* Poll shared memory every 10ms for responsive LED updates */
        Cy_SysLib_Delay(POLLING_DELAY_MS);
    }
}
```

main.c file for PPCA CPU Core 0

```
/*******************************************************************************
* Header Files
*******************************************************************************/

#include "cy_pdl.h"
#include "cycfg.h"
#include <stdio.h>

/*******************************************************************************
* Macros
*******************************************************************************/
/* Toggle period: 1 second */
#define LOOP_DELAY_MS 1000

/* Shared memory addresses in M4 shared memory space (0x20040000-0x20043FFF) */
/* All cores can access M4 shared memory for inter-core communication */
#define PPCA_CPU0_M4_VAR_ADDRESS 0x20040400  /* Used by this core (CPU0) */


/*******************************************************************************
* Global Variables
*******************************************************************************/


/*******************************************************************************
* Function Prototypes
*******************************************************************************/


/*******************************************************************************
* Function Name: main
********************************************************************************
* Summary:
* PPCA Core 0 application:
*    - Toggles shared variable between 0 and 1 every 1 second
*    - Main core monitors this variable and controls LED1
*    - LED1 state reflects this core's activity
*
* Parameters:
*  void
*
* Return:
*  int
*
*******************************************************************************/
int main(void)
{
    /* Pointer to shared memory location for inter-core communication */
    uint32_t *shared_var = (uint32_t *)PPCA_CPU0_M4_VAR_ADDRESS;

    /* Initialize shared variable to 0 */
```

```
        *shared_var = 0;

        /* Main loop: Toggle shared variable every 1 second */
        for(;;)
        {
            /* Wait for 1 second */
            Cy_SysLib_Delay(LOOP_DELAY_MS);

            /* Toggle between 0 and 1 (logical NOT converts 0->1 and 1->0) */
            *shared_var = !(*shared_var);
        }
    }
```

main.c file for PPCA CPU Core 1

```
/*******************************************************************************
* Header Files
*******************************************************************************/

#include "cy_pdl.h"
#include "cycfg.h"
#include <stdio.h>

/*******************************************************************************
* Macros
*******************************************************************************/
/* Toggle period: 1 second */
#define LOOP_DELAY_MS 1000

/* Shared memory addresses in M4 shared memory space (0x20040000-0x20043FFF) */
/* All cores can access M4 shared memory for inter-core communication */
#define PPCA_CPU1_M4_VAR_ADDRESS 0x20040800  /* Used by this core (CPU1) */

/*******************************************************************************
* Global Variables
*******************************************************************************/


/*******************************************************************************
* Function Prototypes
*******************************************************************************/


/*******************************************************************************
* Function Name: main
********************************************************************************
* Summary:
* PPCA Core 1 application:
*    - Toggles shared variable between 0 and 1 every 1 second
*    - Starts with 500ms offset from Core 0 for staggered LED pattern
*    - Main core monitors this variable and controls LED2
*    - LED2 state reflects this core's activity
*
* Parameters:
*  void
*
* Return:
*  int
*
*******************************************************************************/
int main(void)
{
    /* Pointer to shared memory location for inter-core communication */
    uint32_t *shared_var = (uint32_t *)PPCA_CPU1_M4_VAR_ADDRESS;

    /* Initialize shared variable to 0 */
```

```
        *shared_var = 0;

        /* Initial delay to stagger Core 1 toggles from Core 0 by 500ms */
        /* This creates a visually appealing alternating LED pattern */
        Cy_SysLib_Delay(500);

        /* Main loop: Toggle shared variable every 1 second */
        for(;;)
        {
            /* Wait for 1 second */
            Cy_SysLib_Delay(LOOP_DELAY_MS);

            /* Toggle between 0 and 1 (logical NOT converts 0->1 and 1->0) */
            *shared_var = !(*shared_var);
        }
    }
```

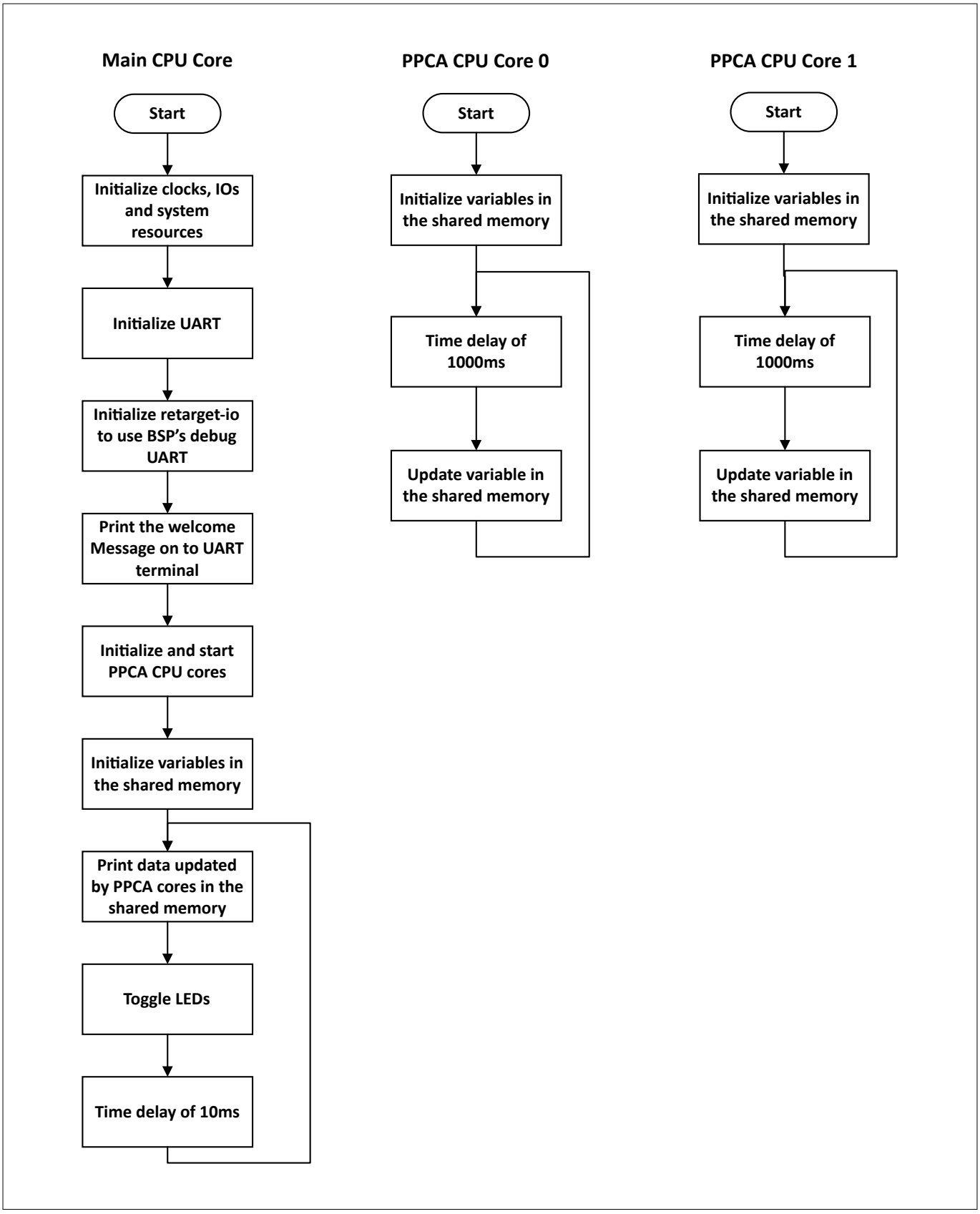


Figure 16 Firmware flowchart

This completes the summary of how the firmware works in the code example. For deeper understanding, refer the source files.

5.4.1.4 Build the application

This section shows how to build the application.

1. Select the parent application project in the Project Explorer view
2. Click Build Project shortcut under the Hello_World group in the Quick Panel

   It selects the build configuration from the Makefile and compiles/links all projects that constitute the application. By default, Debug configurations are selected
3. The Console view lists the results of the build operation, as Figure 17 shows

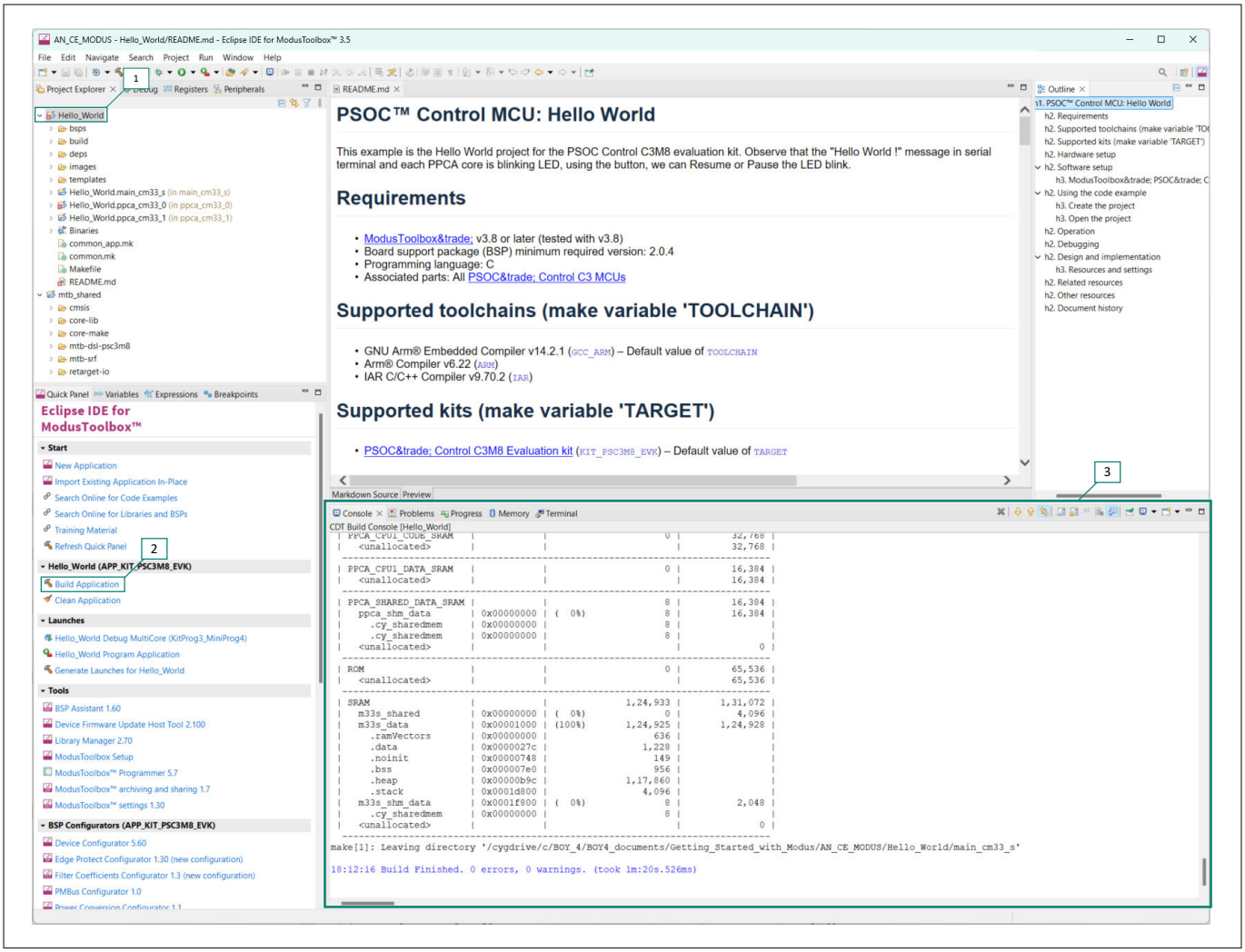


Figure 17 Build the application

If you encounter errors, revisit the earlier steps to ensure that you completed all the required tasks.

You can also use the command-line interface (CLI) to build the application. See the Build system section in the ModusToolbox™ tools package user guide. This document is located in the/docs_<version>/folder in the ModusToolbox™ installation.

5.4.1.5 Program the device

This section shows how to program the PSC3M/P8, PSC3M/P7 MCU.

- Install the ModusToolboxProgTools from ModusToolbox™ setup

ModusToolbox™ software uses the OpenOCD protocol to program and debug applications on PSOC™ Control C3M8 on the evaluation kit.

- As the evaluation kit is with a built-in programmer, connect the board to your computer using the provided USB cable

If you are developing on your own hardware, you may need a hardware programmer/debugger, for example, MiniProg.

To program the application,

1. Connect to the board and perform the following
2. Select the application project and click on the Hello_World Program Application shortcut under the Launches group in the Quick Panel, as Figure 18 shows. The IDE will select and run the appropriate run configuration. Note that this step will also perform a build if any files have been modified since the last build

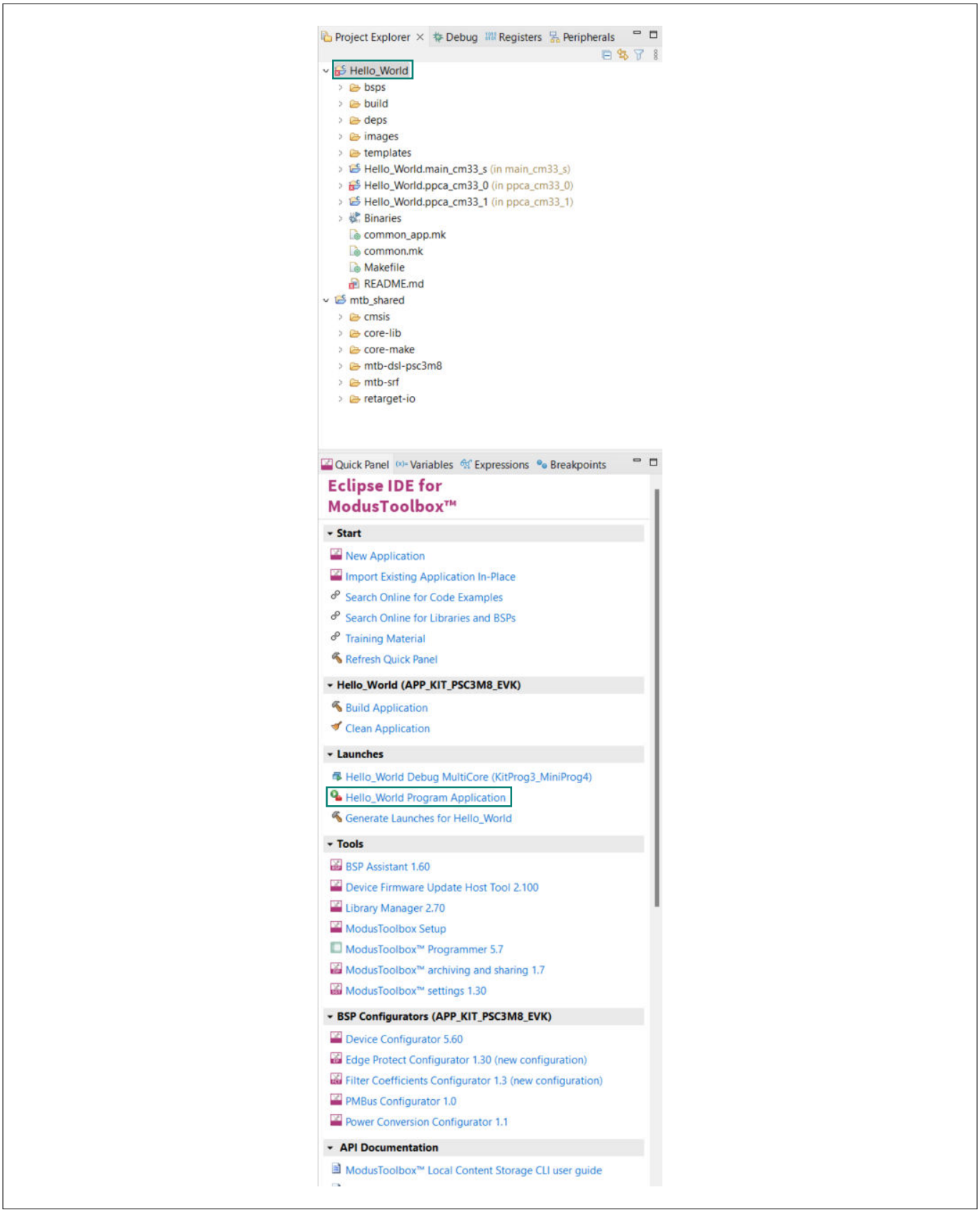


Figure 18 Programming an application to a device

The Console view lists the results of the programming operation, as shown in Figure 19.

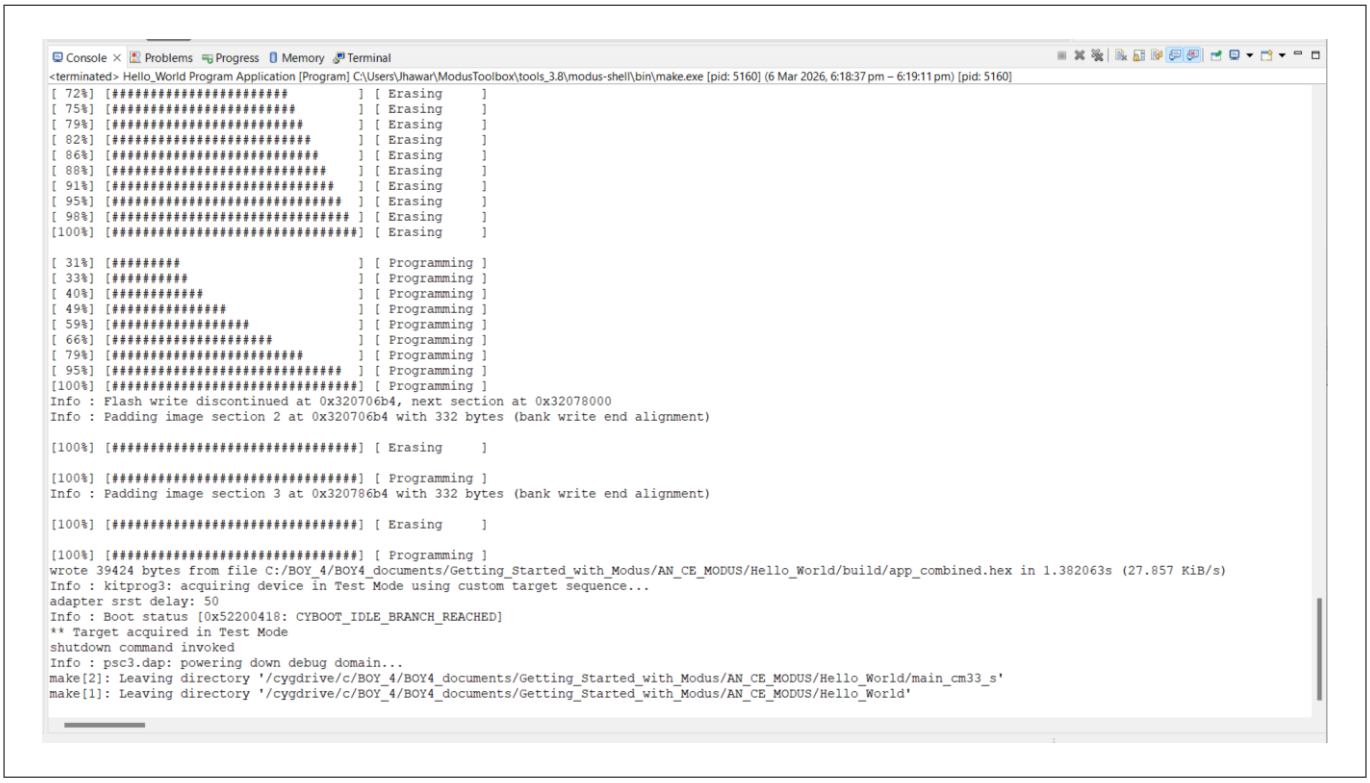


Figure 19 Console – programming results

5.4.1.6 Test your design

This section describes how to test your design.

Follow these steps to observe the output of your design. This note uses Tera Term as the UART terminal emulator to view the results, but you can use any terminal of your choice to view the output.

1. Select the serial port

Launch Tera Term and select the USB-UART COM port as Figure 20 shows. Note that your COM port number may be different

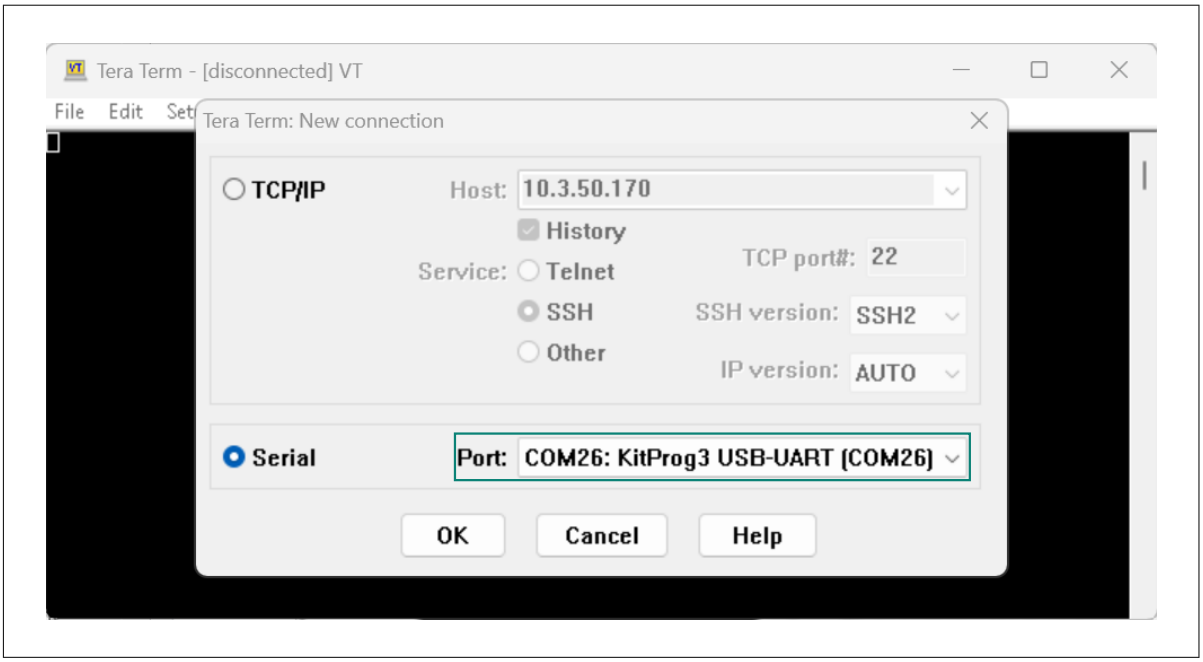


Figure 20 Selecting the KitProg3 COM port in Tera Term

2. Set the baud rate

Set the baud rate to 115200 under **Setup** > **Serial port** as Figure 21 shows

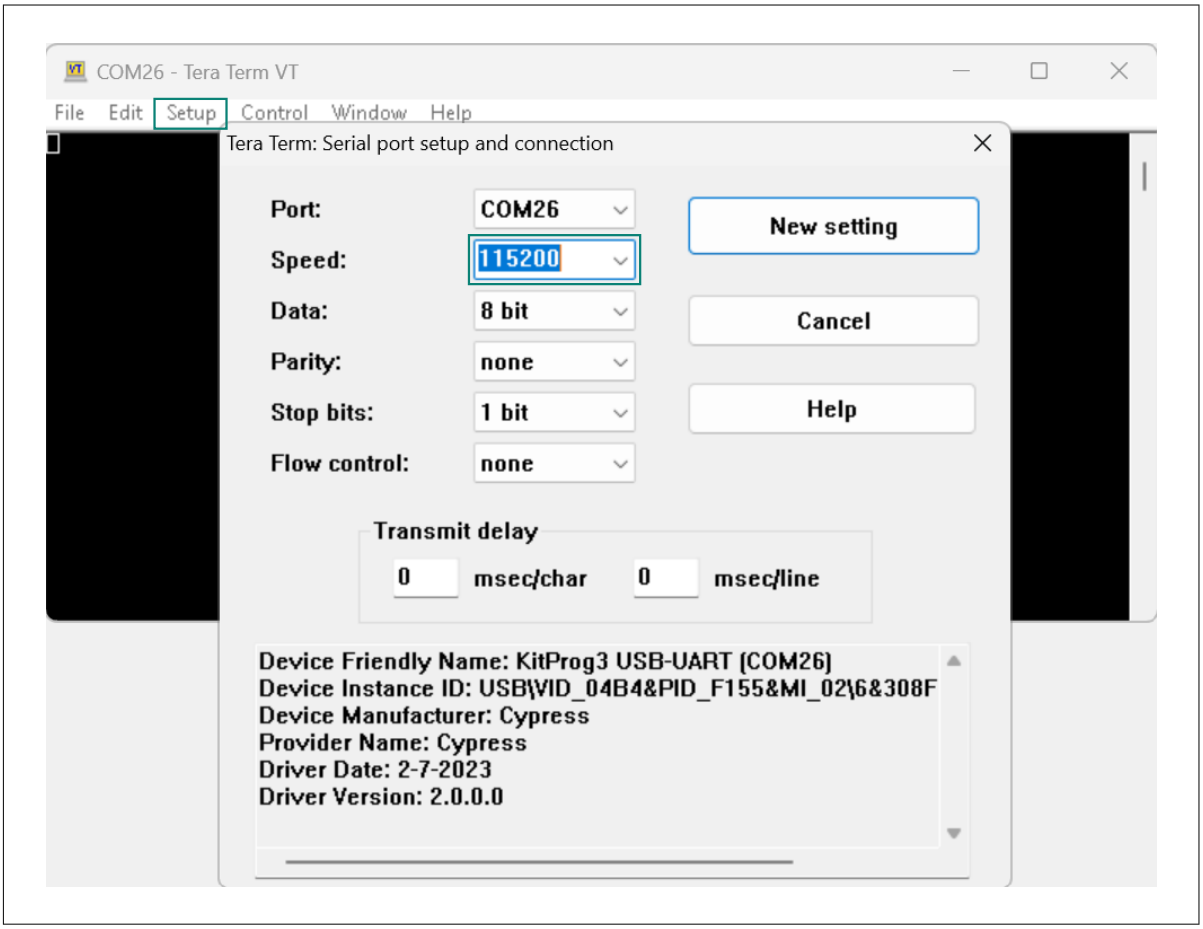


Figure 21 Configuring the baud rate in Tera Term

3. Reset the device

Press the reset switch (SW1) on the kit. The message shown in Figure 22 appears on the terminal. The user LEDs on the kit will start blinking. LED1 for PPCA core1, LED 2 for PPCA core 2 and LED3 for main core

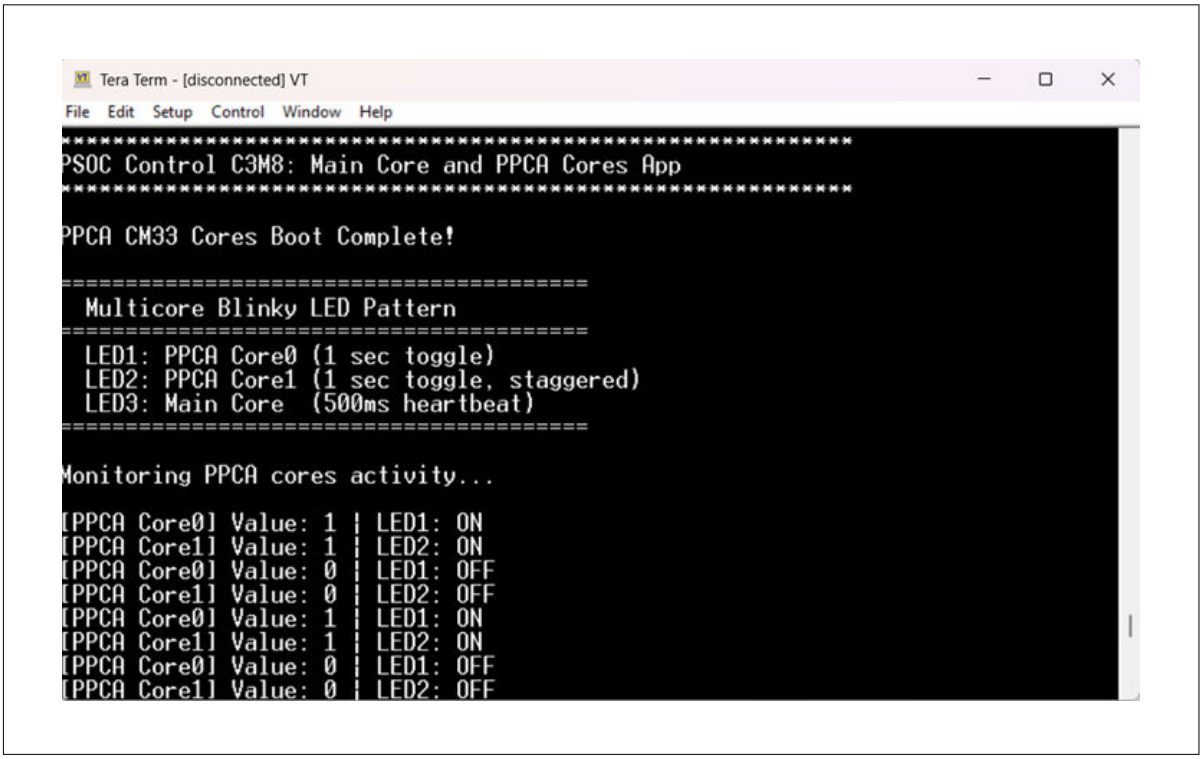


Figure 22 Printed UART message on reset

Here in this example each PPCA core toggles a shared memory variable, and the main core monitors these variables to control LED1, LED2, and LED3 and print status messages via UART, creating a synchronized LED blinky pattern across all three cores.

5.4.1.7 Debugging the application using KitProg3/MiniProg4

PSC3M/P8, PSC3M/P7 kits come with KitProg3 onboard programmer/debugger. See the KitProg3 user guide for details of KitProg3.

The Eclipse IDE contains several launch configurations that control various settings for programming the devices and launching the debugger. Depending on the kit and the type of applications you are using, there are various launch configurations available. One such configuration is the KitProg3/MiniProg4 launch configuration. Refer to the "PSOC™ MCU programming/debugging" section in the Eclipse IDE for ModusToolbox™user guide for more details on the launch configurations.

When an application is created, the tool generates the launch configurations for KitProg3_MiniProg4 or J-link under Launches in the Quick Panel. For the PSC3M/P8, PSC3M/P7 Evaluation Kit, it will generate launch configurations for KitProg3, as shown in the Figure 23.

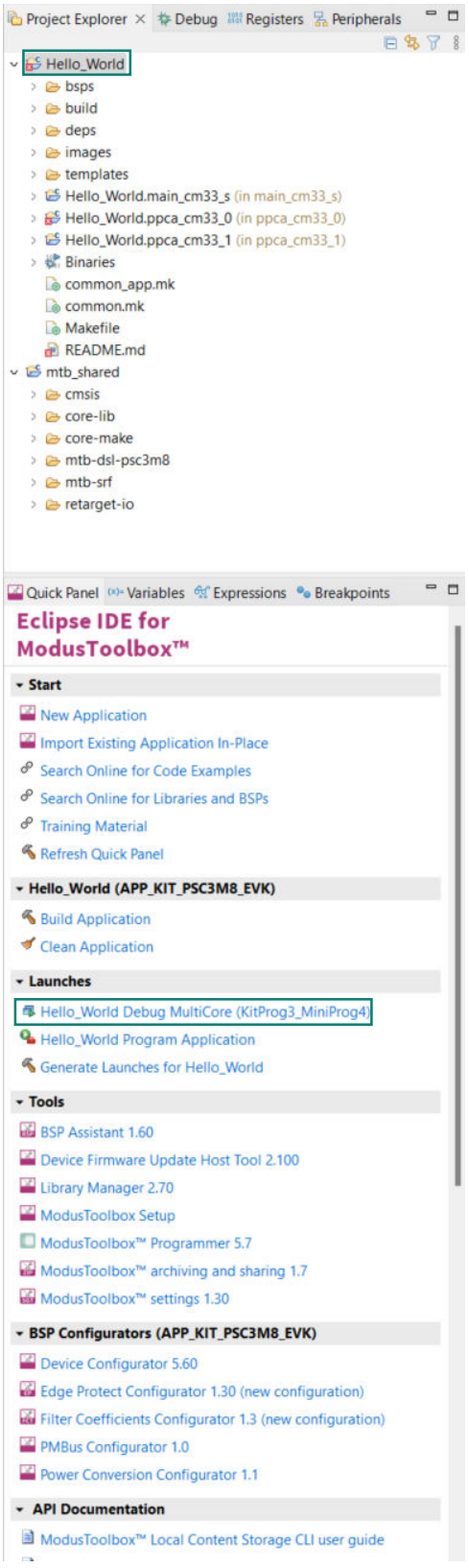


Figure 23 KitProg3/MiniProg4 launch debug configuration

Connect the device to the host machine and click on the Hello_World Debug MultiCore (KitProg3_MiniProg4) launch to start debugging, as shown in Figure 23. Once the debugging starts, the execution halts at the main() function, and the user can start debugging from the start of main(), as shown in the following Figure 24.

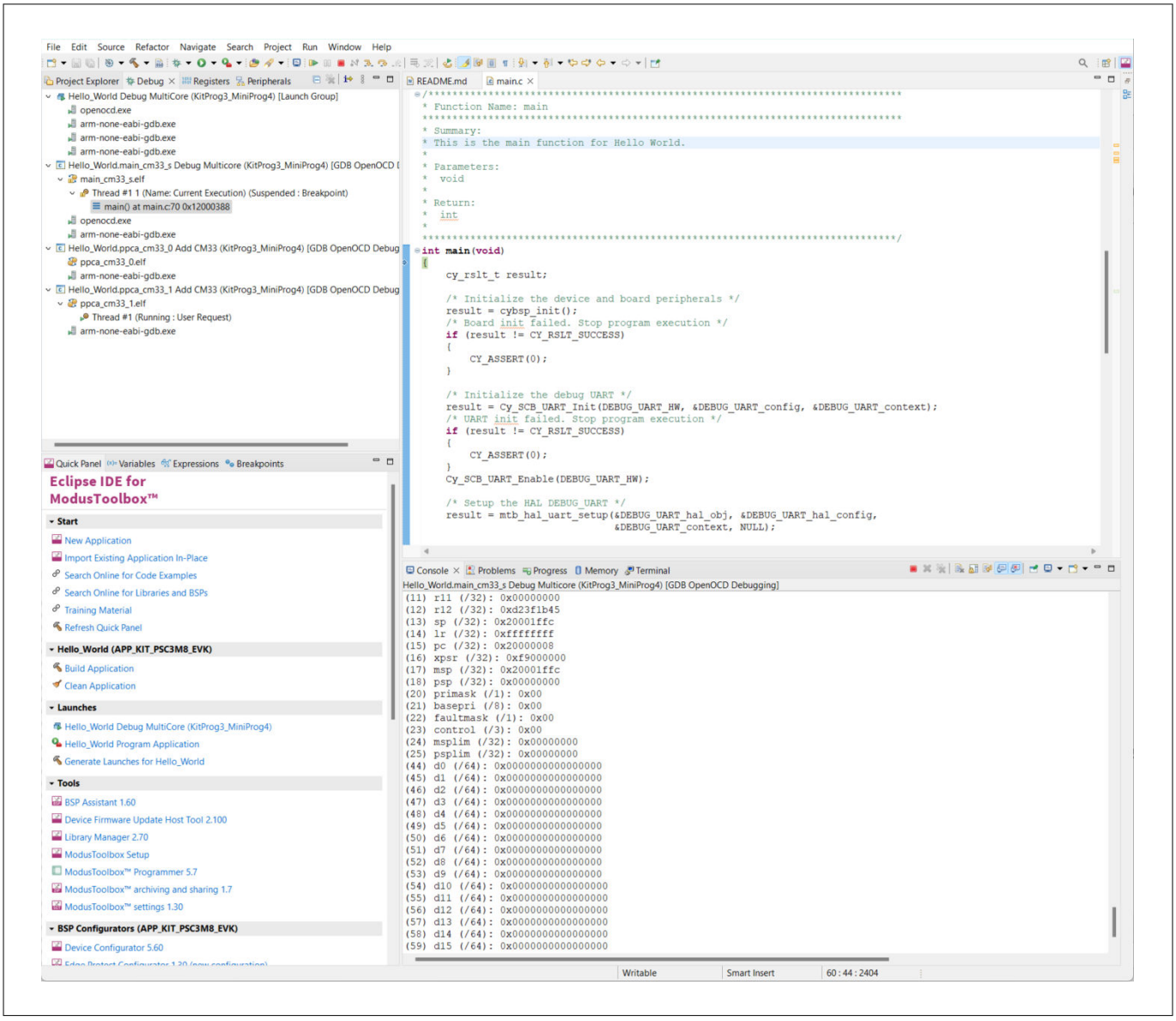


Figure 24 Debugging main()

5.4.2 Visual Studio Code (VS Code) for ModusToolbox™

Refer to the Visual Studio Code for ModusToolbox™ user guide for creating a new application on VS Code.

5.4.3 IAR Embedded Workbench for ModusToolbox™ user guide

Refer to the IAR Embedded Workbench for ModusToolbox™ user guide for creating a new application on IAR Embedded workbench.

5.4.4 Keil µVision for ModusToolbox™ user guide

Refer to the Keil µVision for ModusToolbox™ user guide for creating a new application on Keil µVision.

5.5 Working with PSC3M/P8, PSC3M/P7 MCU devices main CPU and PPCA CPUs

As shown in the section Architecture and product lines, the PSC3M/P8, PSC3M/P7 devices contains three Cortex® M33 CPUs, one main core, and two cores inside the PPCA subsystem. To make use of all the CPU cores, follow the multicore project structure for firmware development. see PSC3M/P8, PSC3M/P7 MCUs multicore application structure.

As the main CPU core is TrustZone®-enabled, it can operate in both secure and non-secure modes. In contrast, the PPCA CPU cores are not TrustZone®-enabled, which means they can only run in non-secure mode. Consequently, we will have four sub-projects in the PSC3M/P8, PSC3M/P7 multicore project. These sub-projects will include: (1) a main core secure application, (2) a main core non-secure application, (3) a PPCA Core 0 non-secure application, and (4) a PPCA Core 1 non-secure application. In three folder structure the non-secure project is not available and other thing remain same.

The MCU execution starts with the main core in secure mode. So, the first application to run is the main core secure application. It sets up the environment for the non-secure application to run and starts the main core non-secure application. PPCA cores will start running only after the main core secure or non-secure application initialized the PPCA Subsystem and PPCA CPU cores and then started it.

The main core can access all peripherals inside the PPCA SS and the PPCA CPU cores have restricted access to the resources outside of the PPCA subsystem which is imposed by the PPC configuration. In addition to that, the address of the resources is different range when accessed from the main CPU and the PPCA CPUs. This section explains the things to take care of while working with multiple cores in the MCU.

5.5.1 Initialization of resources

One important point we need to be aware of is the initialization of resources. Initialization of clock, clock dividers, trigger multiplexer, and pins should happen in both secure and non-secure apps running on the main CPU. This helps to make the required resources available in the apps required to use it. Initialization of this can be done by calling `cybsp_init()` in both secure app and non-secure app.

As the main core secure app runs first and sets up the non-secure app, only the initialization of the peripherals accessed by the secure app should be done inside it. It helps to separate secure and non-secure apps. Remaining peripherals can be initialized in the non-secure app.

Care must be taken when initializing the PPCA resources. Initializing and accessing of the PPCA resources can be done by both Main CPU and PPCA CPUs. It is advised that the initialization should be done by the main CPU, so that the limited PPCA memory from which the PPCA CPUs are run can be saved.

When it comes to secure and non-secure main CPU apps, as the PPCA CPUs have only the non-secure apps, the initialization and starting of the PPCA CPUs should be done in the main CPU.

5.5.2 Communication between CPUs

We have multiple options for the communication between CPUs and synchronization purposes. The shared memory is the most efficient option available for it. The shared memory is in the PPCA Subsystem, and it is accessible to all three CPUs available in the MCU. Structures can be placed in the shared memory, and it can be used for sharing data between different CPU cores and synchronization.

IPC is available in the system for communication and synchronization. Even though it is an easy mechanism available, it comes with its overhead and delay.

6 Summary

This application note explored the PSC3M/P8, PSC3M/P7 MCU devices architecture and the associated development tools. PSC3M/P8, PSC3M/P7 devices are a truly programmable embedded system-on-chip with configurable analog and digital peripheral functions, memory, and a powerful processor on a chip. The integrated features and low-power modes make PSC3M/P8, PSC3M/P7 devices an ideal choice for power stage converters, industrial drives, smart appliances, and other related applications.

References

Sample Code

[1] Infineon Technologies AG: *PSC3M/P8, PSC3M/P7 code examples*; Available online

Documentation

[2] Infineon Technologies AG: *PSC3M/P8, PSC3M/P7-related documents*; Available online

Glossary

This section lists the most commonly used terms that you might encounter while working with PSOC™ family of devices.

- **Board support package (BSP):** A BSP is the layer of firmware containing board-specific drivers and other functions. The board support package is a set of libraries that provide firmware APIs to initialize the board and provide access to board level peripherals.
- **Peripheral driver library (PDL):** The PDL simplifies software development for the PSC3M/P8, PSC3M/P7 architecture. The PDL reduces the need to understand register usage and bit structures, so easing software development for the extensive set of peripherals available.
- **Hardware abstraction layer (HAL):** The HAL wraps the lower-level drivers (like MTB-PDL-CAT1) and provides a high-level interface to the MCU. The interface is abstracted to work on any MCU
- **Personality:** A personality expresses the configurability of a resource for a functionality. For example, the SCB resource can be configured to be an UART, SPI, or I2C personalities.
- **Middleware:** Middleware is a set of firmware modules that provide specific capabilities like motor control, power conversion, communication networks, etc. to an application. A middleware may be a wrapper on top of the low-level device driver libraries intended to demonstrate a particular feature or application by further abstracting the hardware level details from the user.
- **ModusToolbox™:** An Eclipse-based embedded design platform for embedded systems designers that provides a single, coherent, and familiar design experience, combining the industry's most deployed Wi-Fi and Bluetooth® technologies, and the lowest power, most flexible MCUs with best-in-class control and sensing.
- **KitProg:** The KitProg is an onboard programmer/debugger with USB-I2C and USB-to-UART bridge functionality. The KitProg is integrated onto most PSOC™ development kits.
- **MiniProg3/MiniProg4:** Programming hardware for development that is used to program PSOC™ devices on your custom board or PSOC™ development kits that do not support a built-in programmer.

Revision history

Document revision	Date	Description of changes
**	2025-07-30	Initial release
*A	2026-03-26	Updated the document from 4 folder structure to 3 folder structure which includes image,code snippets and sentences
*B	2026-09-18	Release to web

Trademarks

The Bluetooth® word mark and logos are registered trademarks owned by Bluetooth SIG, Inc., and any use of such marks by Infineon is under license.

PSOC™, formerly known as PSoC™, is a trademark of Infineon Technologies. Any references to PSoC™ in this document or others shall be deemed to refer to PSOC™.

Trademarks

All referenced product or service names and trademarks are the property of their respective owners.

Edition 2026-09-18

Published by

Infineon Technologies AG

81726 Munich, Germany

© 2026 Infineon Technologies AG

All Rights Reserved.

Do you have a question about any aspect of this document?

Email: erratum@infineon.com

Document reference
IFX-sxf1740636095180

Important notice

Products which may also include samples and may be comprised of hardware or software or both ("Product(s)") are sold or provided and delivered by Infineon Technologies AG and its affiliates ("Infineon") subject to the terms and conditions of the frame supply contract or other written agreement(s) executed by a customer and Infineon or, in the absence of the foregoing, the applicable Sales Conditions of Infineon. General terms and conditions of a customer or deviations from applicable Sales Conditions of Infineon shall only be binding for Infineon if and to the extent Infineon has given its express written consent.

For the avoidance of doubt, Infineon disclaims all warranties of non-infringement of third-party rights and implied warranties such as warranties of fitness for a specific use/purpose or merchantability. Infineon shall not be responsible for any information with respect to samples, the application or customer's specific use of any Product or for any examples or typical values given in this document.

If this document is marked as 'Preliminary', 'Target', 'Draft' or 'Proposal', Infineon reserves the right to change all information given in this document at any time without notice. Subject to the development and release of a Product for series supply by Infineon, the technical specifications of the Product are set forth in the relevant datasheet provided by Infineon without such marking.

The data contained in this document is exclusively intended for technically qualified and skilled customer representatives. It is the responsibility of the customer to evaluate the suitability of the Product for the intended application and the customer's specific use and to verify all relevant technical data contained in this document in the intended application and the customer's specific use. The customer is responsible for properly designing, programming, and testing the functionality and safety of the intended application, as well as complying with any legal requirements related to its use.

Unless otherwise explicitly approved by Infineon, Products may not be used in any application where a failure of the Products or any consequences of the use thereof can reasonably be expected to result in personal injury.

However, the foregoing shall not prevent the customer from using any Product in such fields of use that Infineon has explicitly designed and sold it for, provided that the overall responsibility for the application lies with the customer. Infineon expressly reserves the right to use its content for commercial text and data mining (TDM) according to applicable laws, e.g. Section 44b of the German Copyright Act (UrhG). If the Product includes security features:

Because no computing device can be absolutely secure, and despite security measures implemented in the Product, Infineon does not guarantee that the Product will be free from intrusion, data theft or loss, or other breaches ("Security Breaches"), and Infineon shall have no liability arising out of any Security Breaches.

If this document includes or references software:

The software is owned by Infineon under the intellectual property laws and treaties of the United States, Germany, and other countries worldwide. All rights reserved. Therefore, you may use the software only as provided in the software license agreement accompanying the software.

If no software license agreement applies, Infineon hereby grants you a personal, non-exclusive, non-transferable license (without the right to sublicense) under its intellectual property rights in the software (a) for software provided in source code form, to modify and reproduce the software solely for use with Infineon hardware products, only internally within your organization, and (b) to distribute the software in binary code form externally to end users, solely for use on Infineon hardware products. Any other use, reproduction, modification, translation, or compilation of the software is prohibited.

For further information on the Product, technology, delivery terms and conditions, and prices, please contact your nearest Infineon office or visit www.infineon.com.