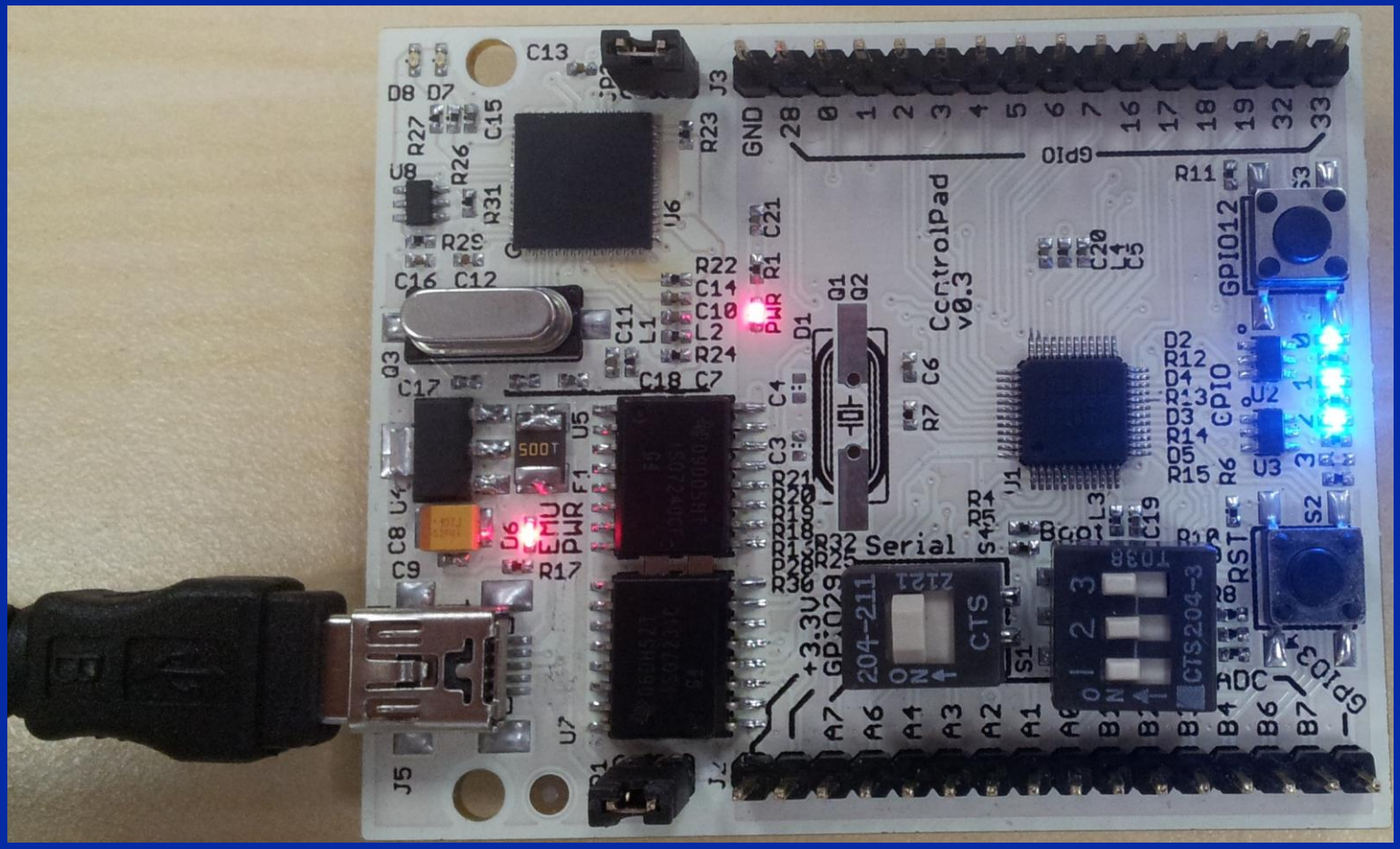
C2000 Microcontroller 1-Day Workshop



Texas Instruments Technical Training

C2000 MCU 1-Day Workshop Outline

- **Workshop Introduction**
- **Architecture Overview**
- **Programming Development Environment**
  - *Lab: Linker command file*
- **Peripheral Register Header Files**
- **Reset, Interrupts and System Initialization**
  - *Lab: Watchdog and interrupts*
- **Control Peripherals**
  - *Lab: Generate and graph a PWM waveform*
- **Flash Programming**
  - *Lab: Run the code from flash memory*
- **The Next Step…**

C2000 MCU 1-Day Workshop Outline

- **Workshop Introduction**
- **Architecture Overview**
- **Programming Development Environment**
  - *Lab: Linker command file*
- **Peripheral Register Header Files**
- **Reset, Interrupts and System Initialization**
  - *Lab: Watchdog and interrupts*
- **Control Peripherals**
  - *Lab: Generate and graph a PWM waveform*
- **Flash Programming**
  - *Lab: Run the code from flash memory*
- **The Next Step…**

Introductions

- Name
- Company
- Project Responsibilities
- DSP / Microcontroller Experience
- TI Processor Experience
- Hardware / Software - Assembly / C
- Interests

TI Embedded Processing Portfolio

TI Embedded Processors

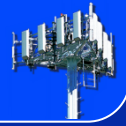
Microcontrollers (MCUs)			ARM®-Based Processors		Digital Signal Processors (DSPs)	
16-bit ultra-low power MCUs	32-bit real-time MCUs	32-bit ARM Cortex™-M3 MCUs	ARM Cortex-A8 MPUs	DSP DSP+ARM	Multi-core DSP	Ultra Low power DSP
MSP430™	**C2000™ Delfino™ Piccolo™**	**Stellaris®** ARM® Cortex™-M3	**Sitara™** ARM® Cortex™-A8 & ARM9	**C6000™ DaVinci™** video processors **OMAP™**	**C6000™**	**C5000™**
Up to 25 MHz	40MHz to 300 MHz	Up to 100 MHz	300MHz to >1GHz	300MHz to >1Ghz +Accelerator	24.000 MMACS	Up to 300 MHz +Accelerator
Flash 1 KB to 256 KB	Flash, RAM 16 KB to 512 KB	Flash 8 KB to 256 KB	Cache, RAM, ROM	Cache RAM, ROM	Cache RAM, ROM	Up to 320KB RAM Up to 128KB ROM
Analog I/O, ADC LCD, USB, RF	PWM, ADC, CAN, SPI, I²C	USB, ENET MAC+PHY CAN, ADC, PWM, SPI	USB, CAN, PCIe, EMAC	USB, ENET, PCIe, SATA, SPI	SRIO, EMAC DMA, PCIe	USB, ADC McBSP, SPI, I²C
Measurement, Sensing, General Purpose	Motor Control, Digital Power, Lighting, Ren. Enrgy	Connectivity, Security, Motion Control, HMI, Industrial Automation	Industrial computing, POS & portable data terminals	Floating/Fixed Point Video, Audio, Voice, Security, Confer.	Telecom T&M, media gateways, base stations	Audio, Voice Medical, Biometrics
$0.25 to $9.00	$1.50 to $20.00	$1.00 to $8.00	$5.00 to $20.00	$5.00 to $200.00	$40 to $200.00	$3.00 to $10.00









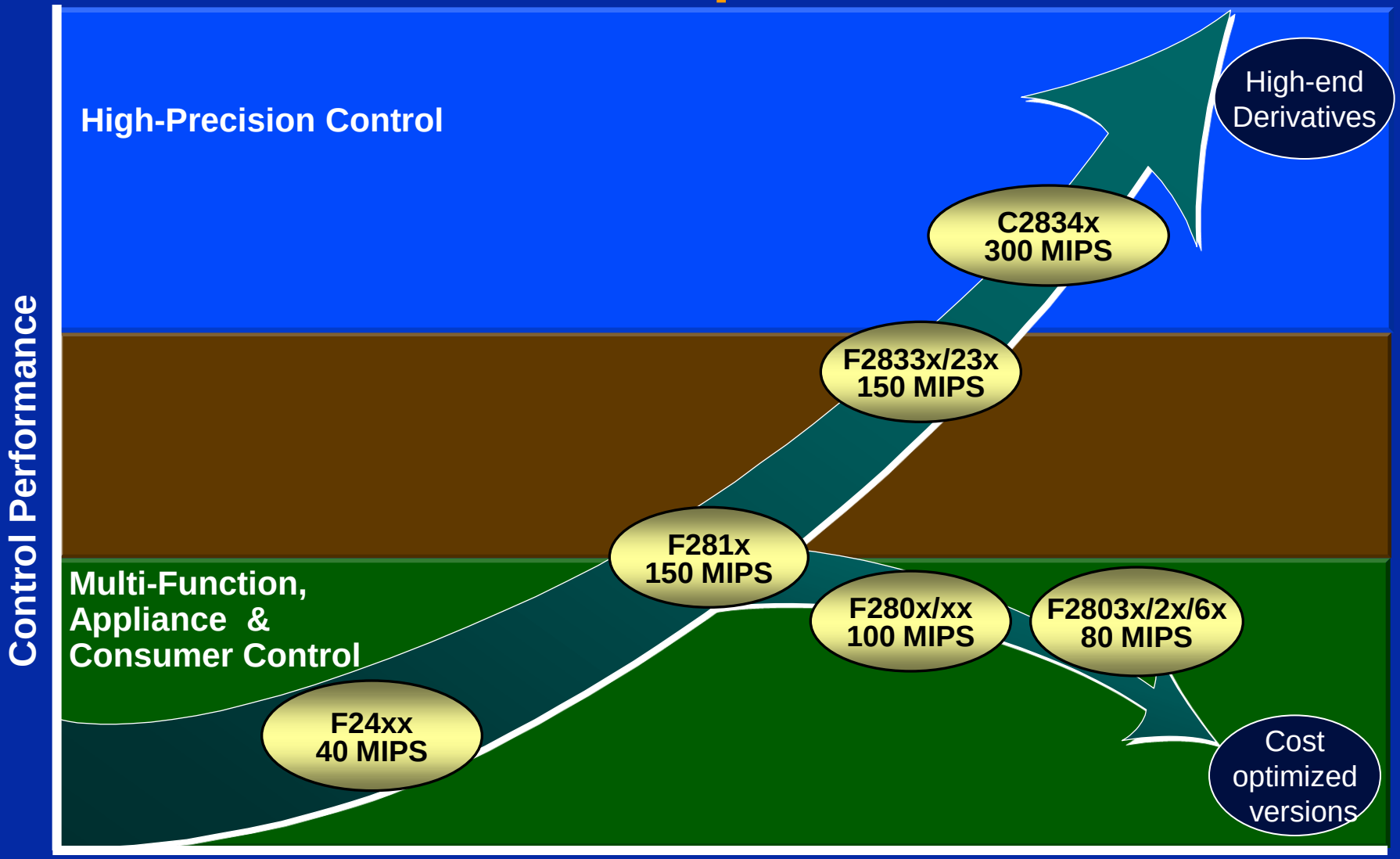


Texas Instruments Developer Network


Software & Dev. Tools


Texas Instruments eXpressDSP™

C2000 Portfolio Expanding with Price/Performance Optimized Derivatives



Broad C2000 Application Base

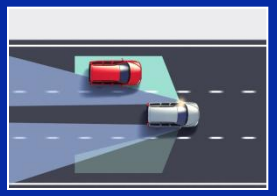


Renewable Energy Generation


Telecom Digital Power



AC Drives, Industrial & Consumer Motor Control

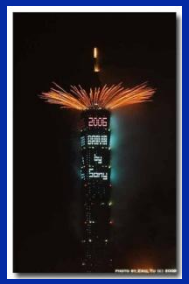
Automotive Radar, Electric Power Steering & Digital Power


Power Line Communications


LED Lighting



Consumer, Medical & Non-traditional








C2000 Delfino / Piccolo Comparison

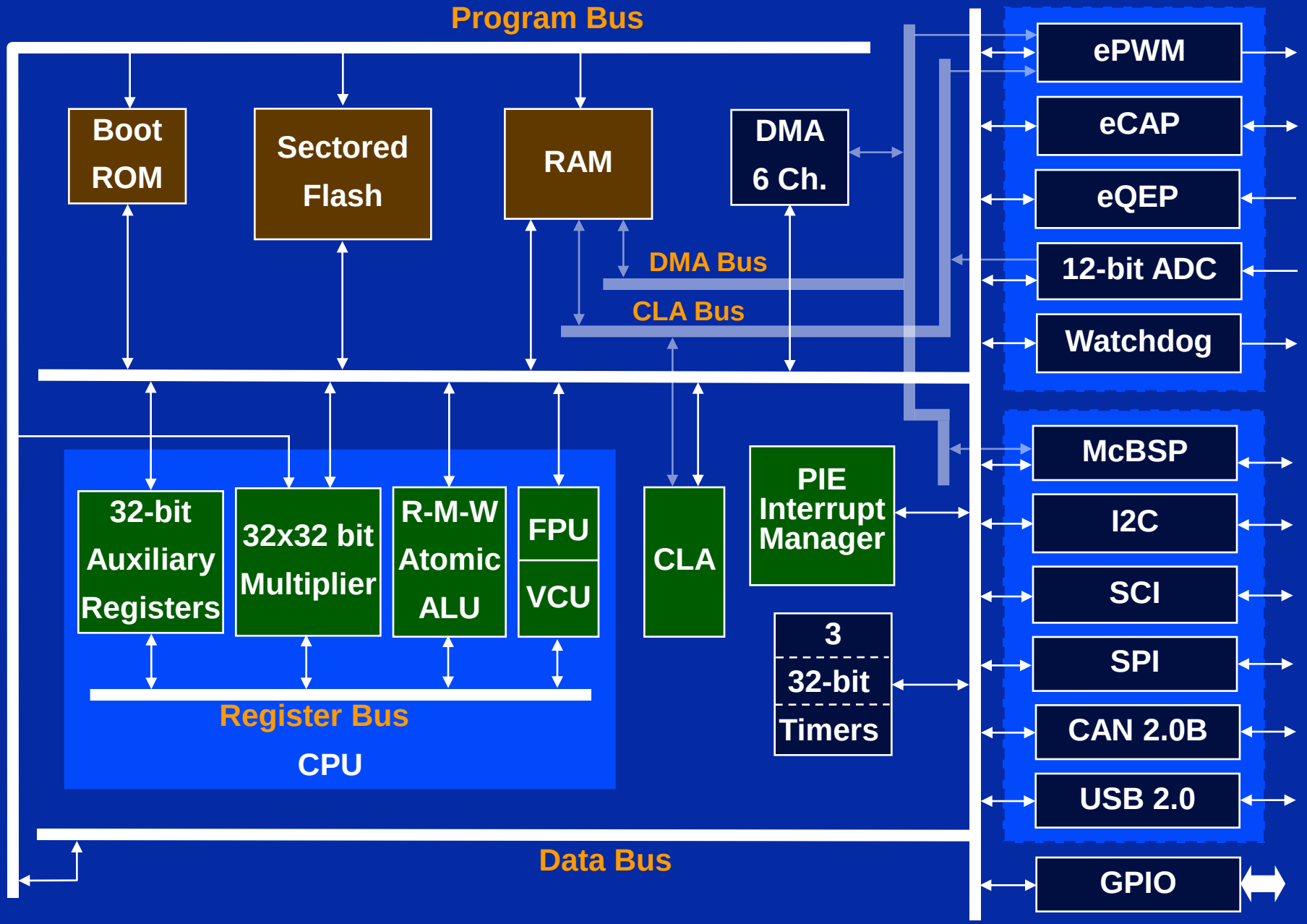
	F2833x	F2803x	F2806x
Clock	150 MHz	60 MHz	80 MHz
Flash / RAM	128Kw / 34Kw	64Kw / 10Kw	128Kw / 50Kw
On-chip Oscillators	-	2	2
VREG / POR / BOR	-	✓	✓
Watchdog Timer	✓	✓	✓
12-bit ADC	SEQ - based	SOC - based	SOC - based
Analog COMP w/ DAC	-	✓	✓
FPU	✓	-	✓
6-Channel DMA	✓	-	✓
CLA	-	✓	✓
VCU	-	-	✓
ePWM / HR ePWM	✓ / ✓	✓ / ✓	✓ / ✓
eCAP / HR eCAP	✓ / -	✓ / -	✓ / ✓
eQEP	✓	✓	✓
SCI / SPI / I2C	✓	✓	✓
LIN	-	✓	-
McBSP	✓	-	✓
USB	-	-	✓
External Interface	✓	-	-

TMS320F28069 controlSTICK

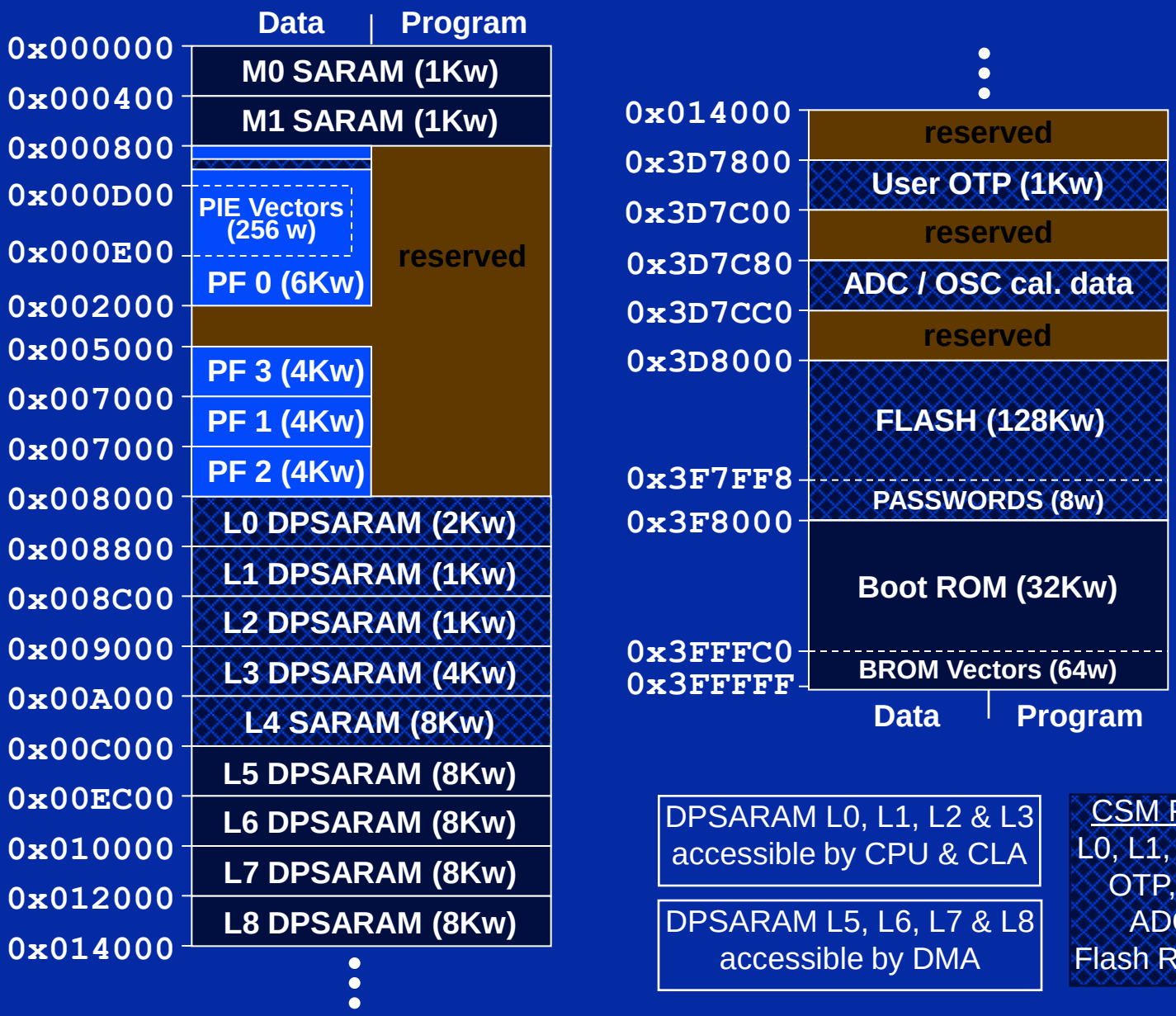
C2000 MCU 1-Day Workshop Outline

- Workshop Introduction
- **Architecture Overview**
- Programming Development Environment
  - *Lab: Linker command file*
- Peripheral Register Header Files
- Reset, Interrupts and System Initialization
  - *Lab: Watchdog and interrupts*
- Control Peripherals
  - *Lab: Generate and graph a PWM waveform*
- Flash Programming
  - *Lab: Run the code from flash memory*
- The Next Step…

TMS320F2806x Block Diagram



TMS320F28069 Memory Map



DPSARAM L0, L1, L2 & L3 accessible by CPU & CLA

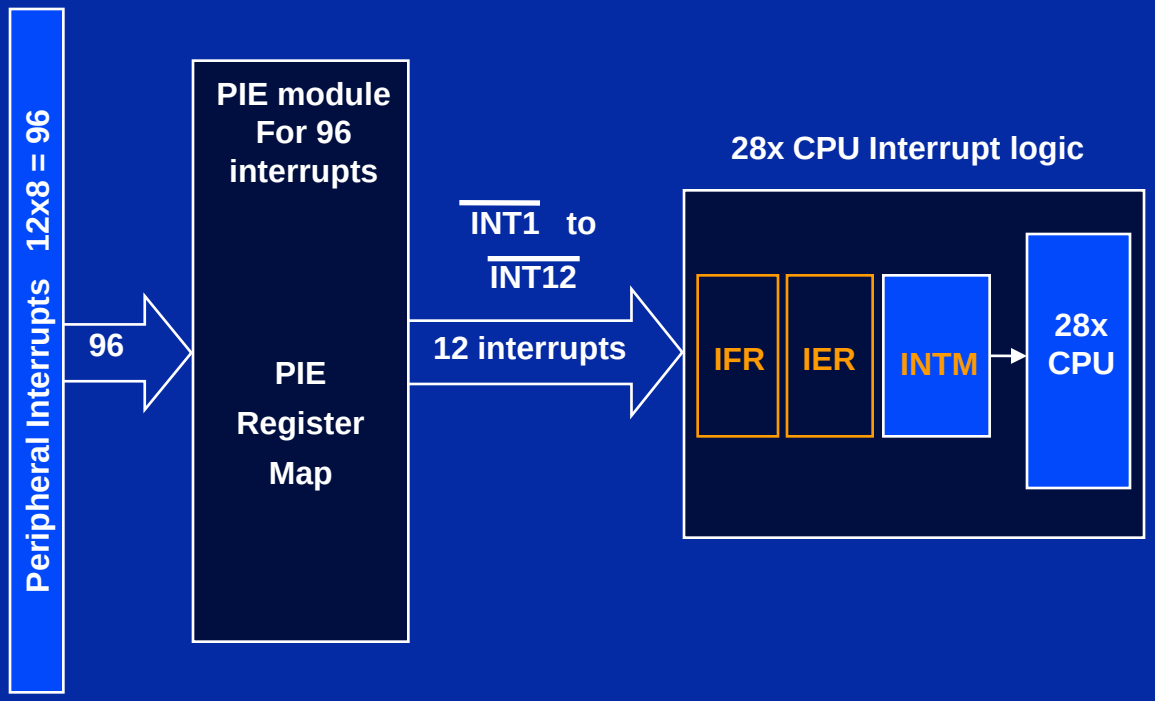
DPSARAM L5, L6, L7 & L8 accessible by DMA

CSM Protected:
L0, L1, L2, L3, L4, OTP, FLASH, ADC CAL, Flash Regs in PF0

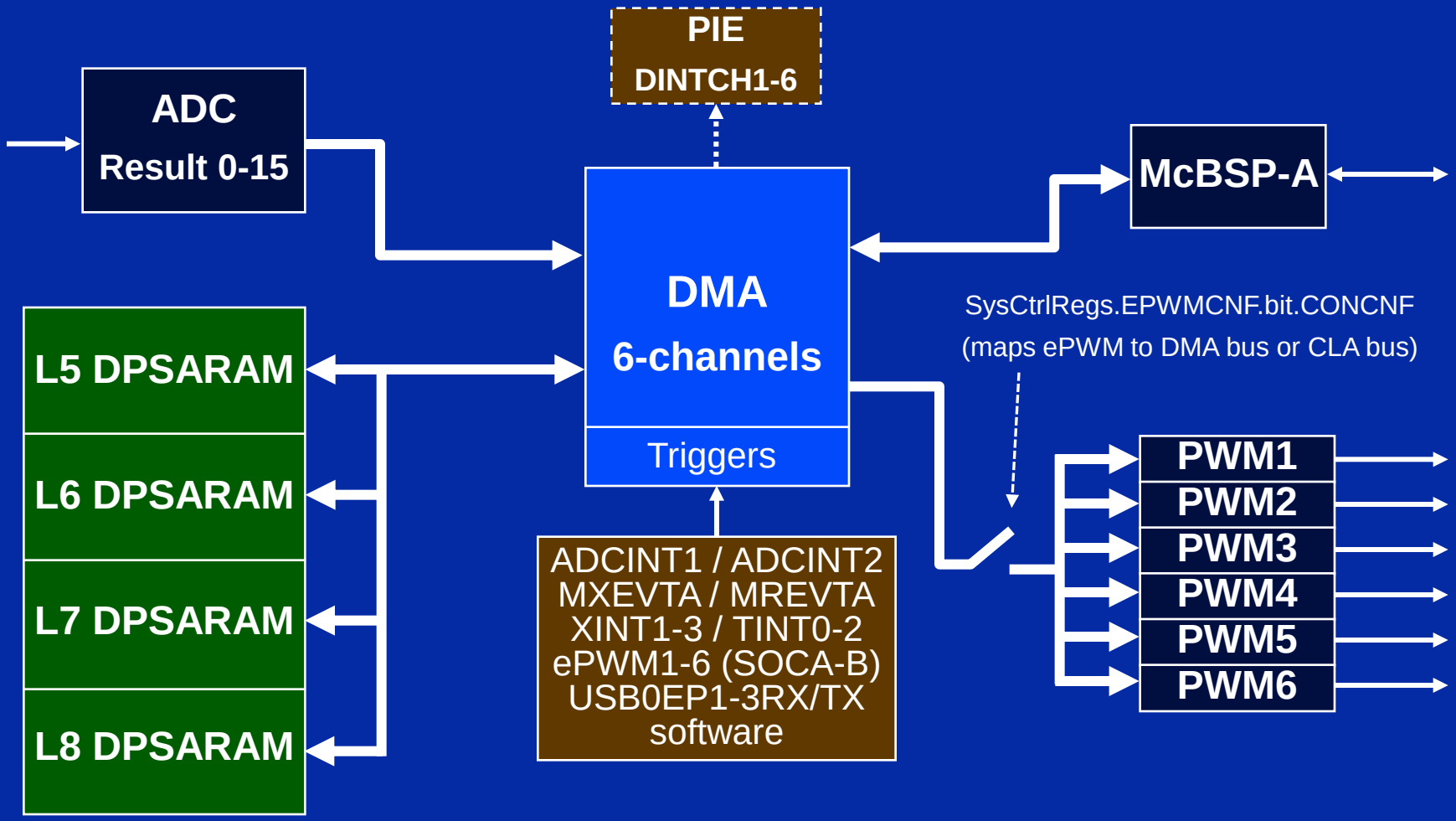
F28x Fast Interrupt Response Manager

- 96 dedicated PIE vectors
- No software decision making required
- Direct access to RAM vectors
- Auto flags update
- Concurrent auto context save

Auto Context Save	
T	ST0
AH	AL
PH	PL
AR1 (L)	AR0 (L)
DP	ST1
DBSTAT	IER
PC(msw)	PC(lsw)

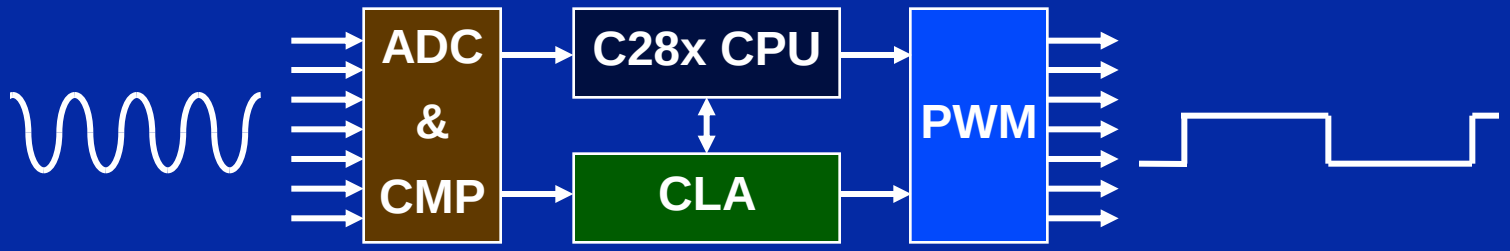


Direct Memory Access (DMA)



Transfers data between peripherals and/or memory *without intervention from the CPU*

Control Law Accelerator (CLA)



- CLA is an independent 32-bit floating-point math accelerator
- Executes algorithms independently and in parallel with the main CPU
- Direct access to ePWM / HRPWM, ADC result and comparator registers
- Responds to peripheral interrupts independently of CPU
- Frees-up CPU for other tasks (communications and diagnostics)

Viterbi, Complex Math, CRC Unit (VCU)

Extends C28x instruction set to support:

- **Viterbi operations**
  - **Decode for communications**
- **Complex math**
  - **16-bit fixed-point complex FFT (5 cycle butterfly)**
    - *used in spread spectrum communications, and many signal processing algorithms*
  - **Complex filters**
    - *used to improve data reliability, transmission distance, and power efficiency*
  - **Power Line Communications (PLC) and radar applications**
- **Cyclic Redundancy Check (CRC)**
  - **Communications and memory robustness checks**

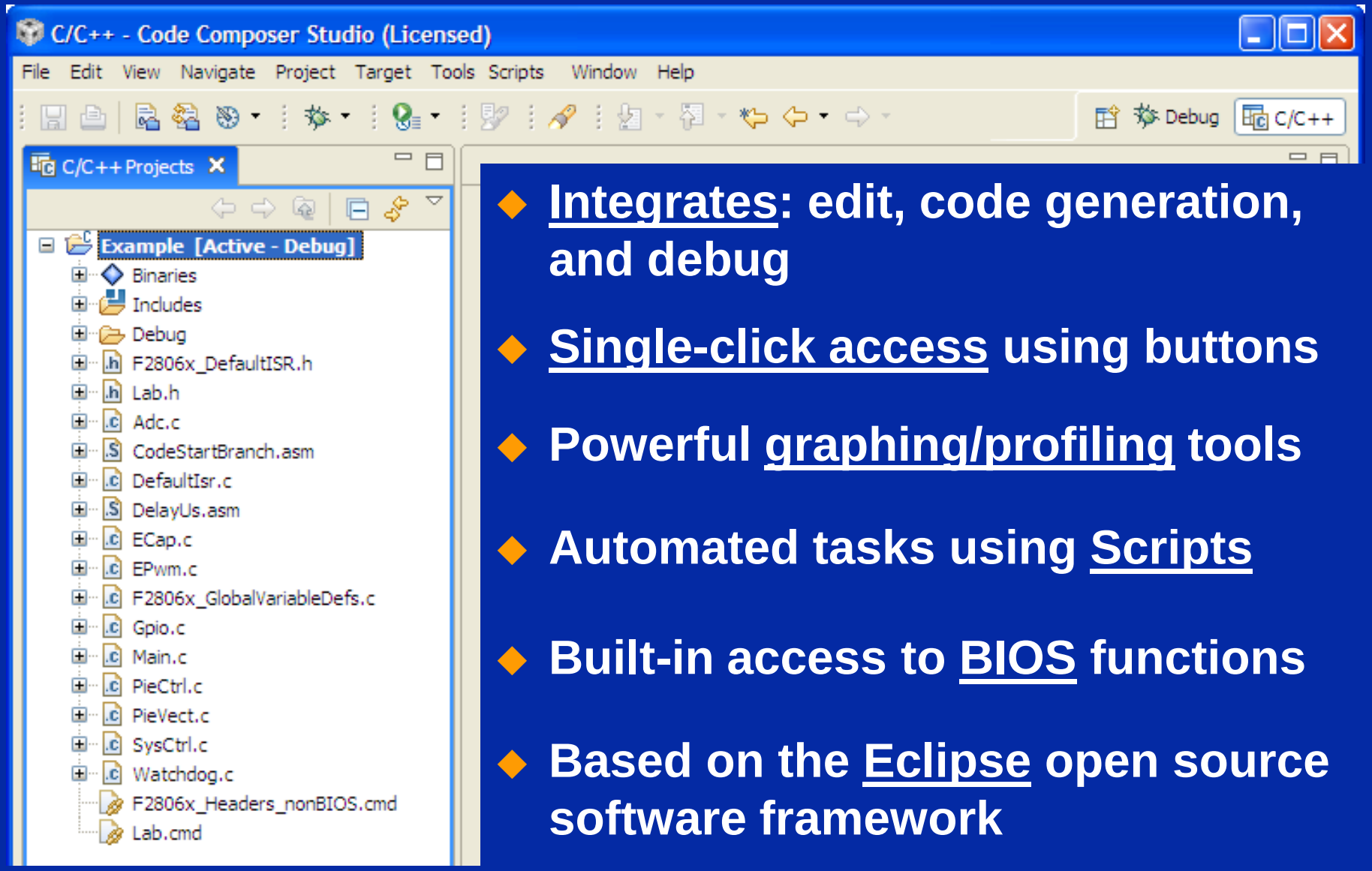
Architecture Summary

- High performance 32-bit CPU
- 32x32 bit or dual 16x16 bit MAC
- IEEE single-precision floating point unit (FPU)
- Hardware Control Law Accelerator (CLA)
- Viterbi, complex math, CRC unit (VCU)
- Atomic read-modify-write instructions
- Fast interrupt response manager
- 128Kw on-chip flash memory
- Code security module (CSM)
- Control peripherals
- 12-bit ADC module
- Comparators
- Direct memory access (DMA)
- Up to 54 shared GPIO pins
- Communications peripherals

C2000 MCU 1-Day Workshop Outline

- **Workshop Introduction**
- **Architecture Overview**
- **Programming Development Environment**
  - *Lab: Linker command file*
- **Peripheral Register Header Files**
- **Reset, Interrupts and System Initialization**
  - *Lab: Watchdog and interrupts*
- **Control Peripherals**
  - *Lab: Generate and graph a PWM waveform*
- **Flash Programming**
  - *Lab: Run the code from flash memory*
- **The Next Step…**

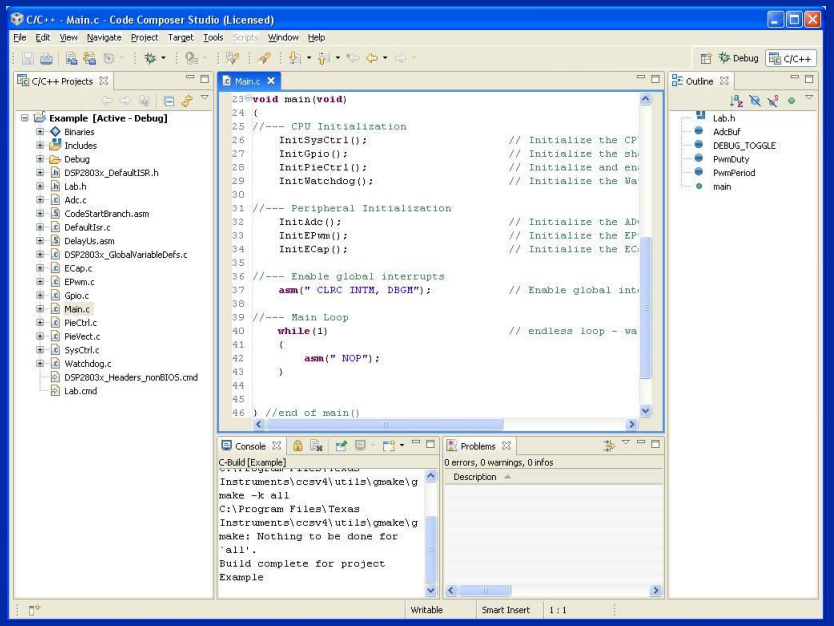
Code Composer Studio: IDE

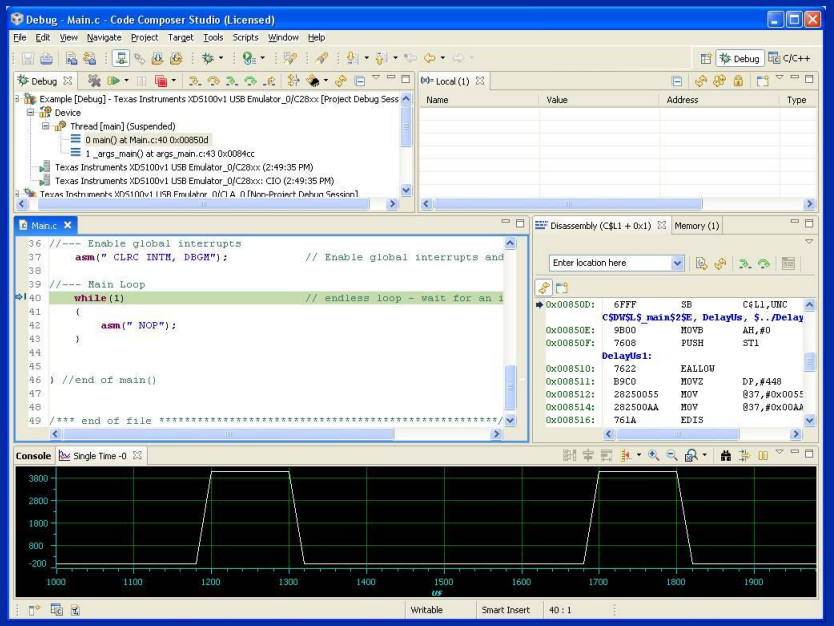


- Integrates: edit, code generation, and debug
- Single-click access using buttons
- Powerful graphing/profiling tools
- Automated tasks using Scripts
- Built-in access to BIOS functions
- Based on the Eclipse open source software framework

C/C++ and Debug Perspective (CCSv4)

- Each perspective provides a set of functionality aimed at accomplishing a specific task





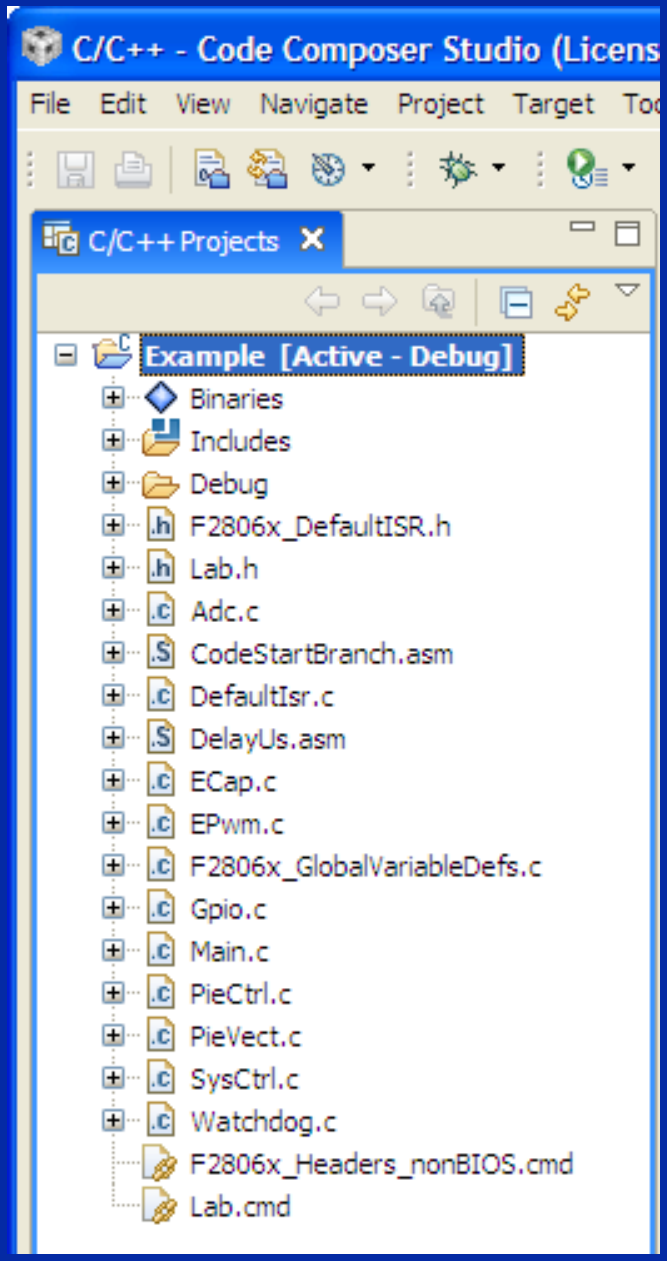
C/C++ Perspective

- Displays views used during code development
  - C/C++ project, editor, etc.

Debug Perspective

- Displays views used for debugging
  - Menus and toolbars associated with debugging, watch and memory windows, graphs, etc.

CCSv4 Project



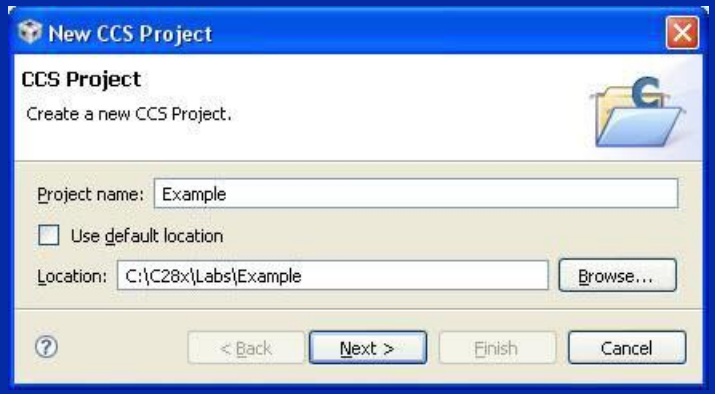
Project files contain:

- **List of files:**
  - Source (C, assembly)
  - Libraries
  - DSP/BIOS configuration file
  - Linker command files
- **Project settings:**
  - Build options (compiler, assembler, linker, and DSP/BIOS)
  - Build configurations

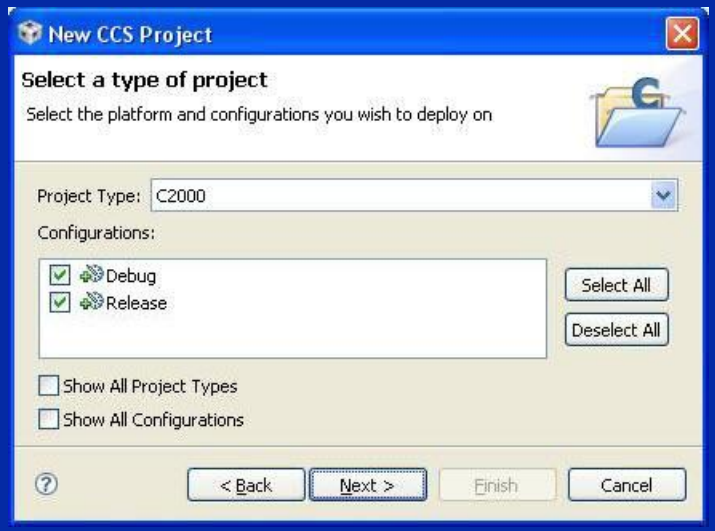
Creating a New CCSv4 Project

- ***File → New → CCS Project***

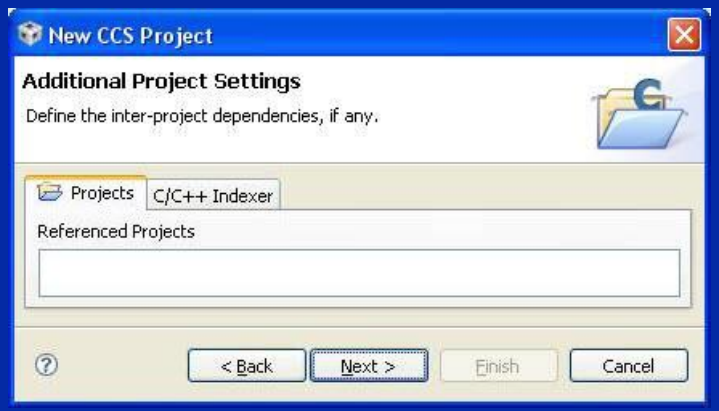
1



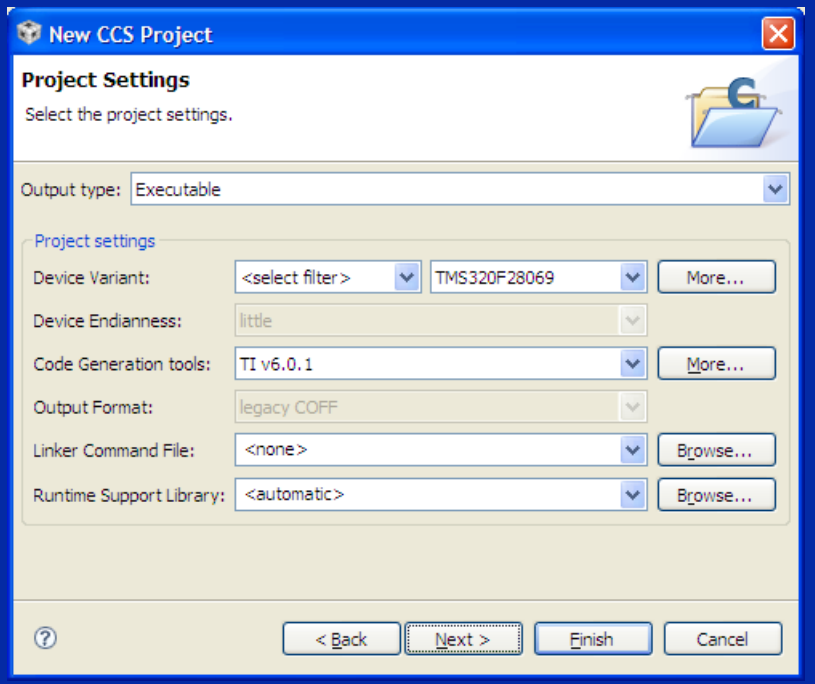
2



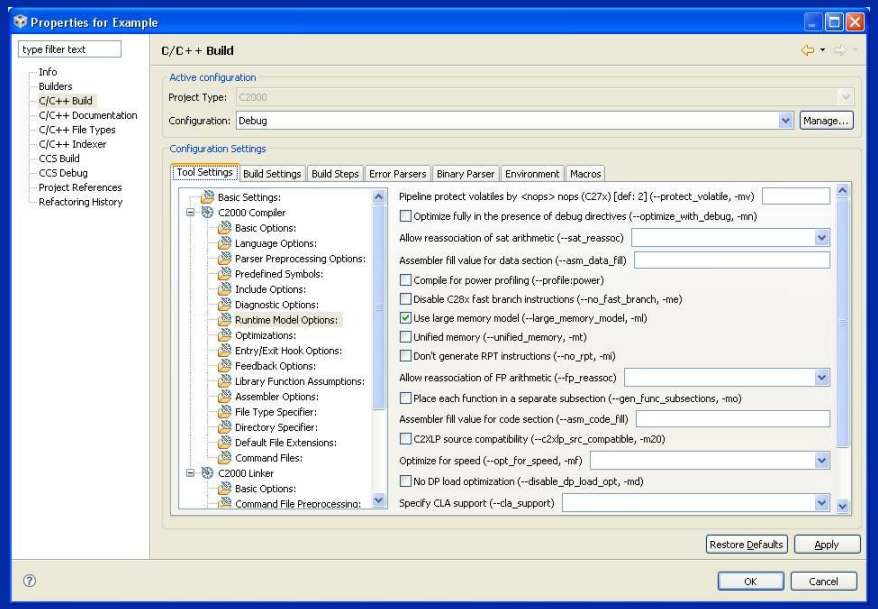
3

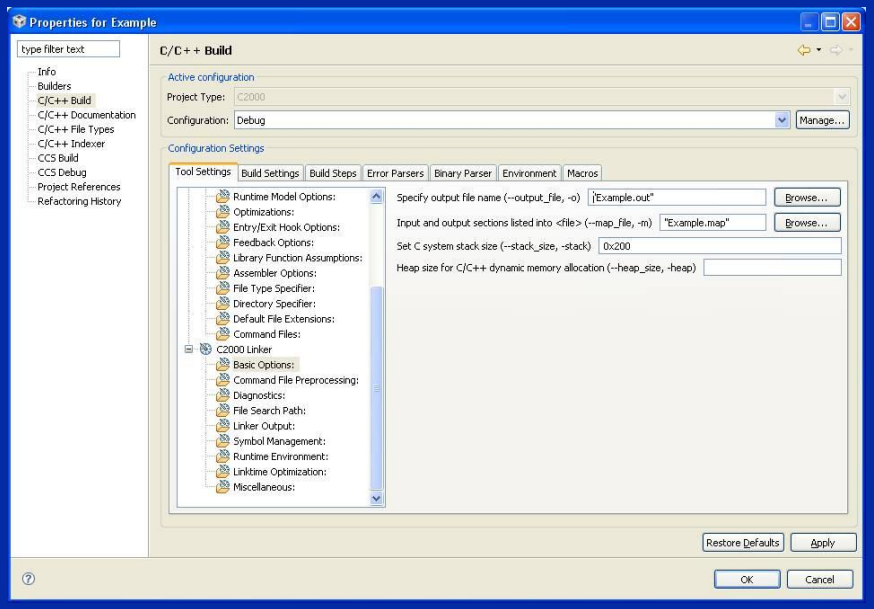


4



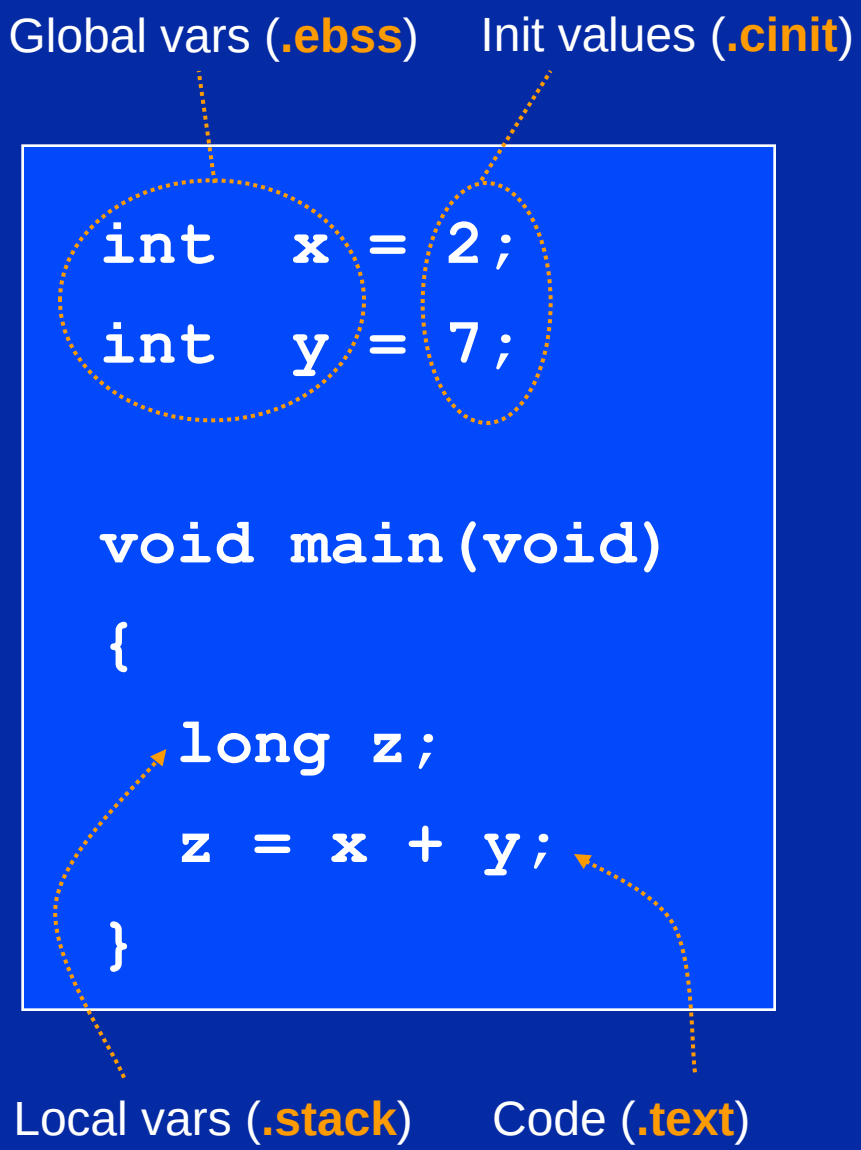
CCSv4 Build Options – Compiler / Linker





- **Compiler**
  - 16 categories for code generation tools
  - Controls many aspects of the build process, such as:
    - Optimization level
    - Target device
    - Compiler / assembly / link options
- **Linker**
  - 9 categories for linking
    - Specify various link options
  - *${PROJECT_ROOT}* specifies the current project directory

Sections



- All code consists of different parts called *sections*
- All default section names begin with “.”
- The compiler has default section names for *initialized* and *uninitialized* sections

Compiler Section Names

Initialized Sections

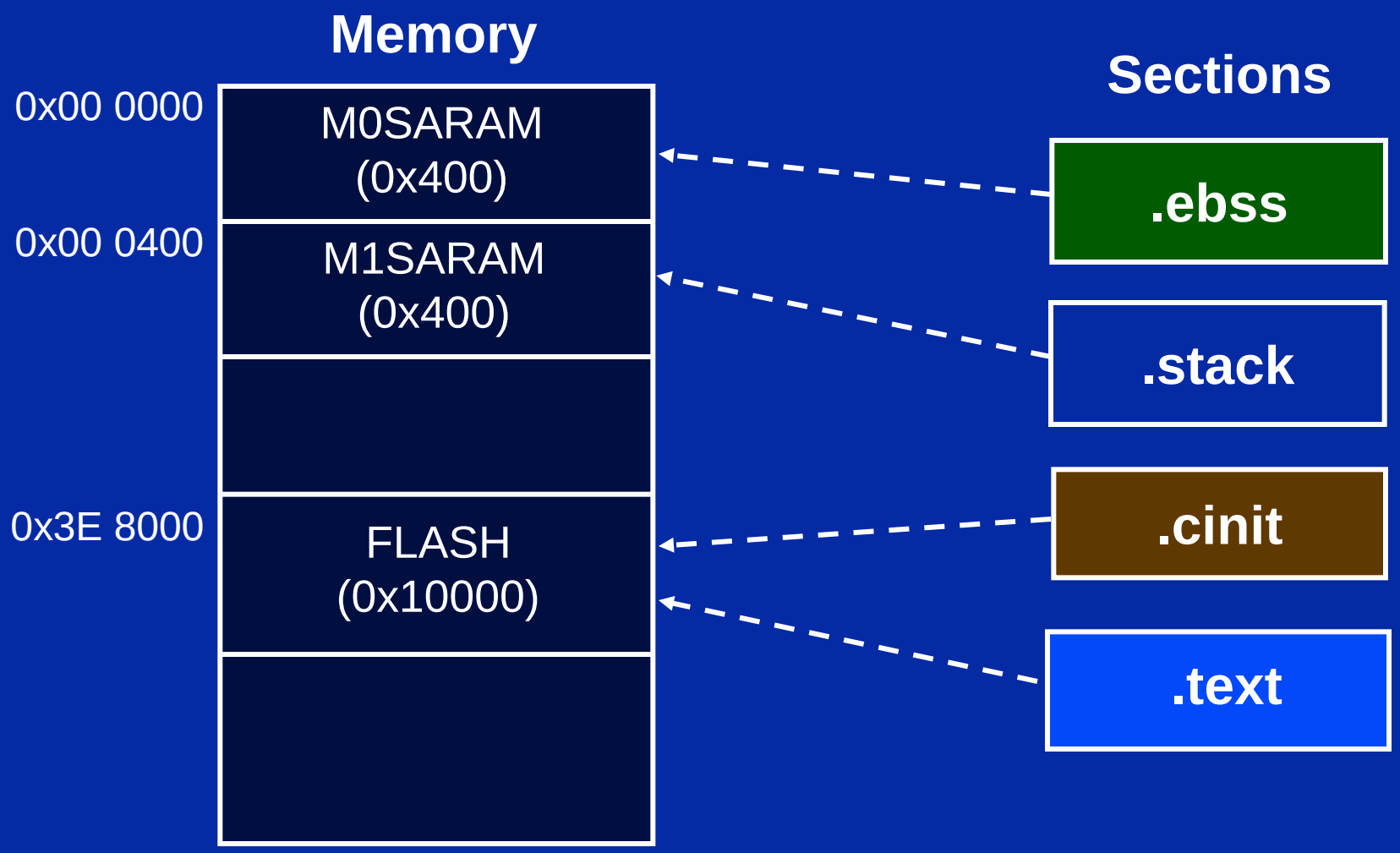
Name	Description	Link Location
.text	code	FLASH
.cinit	initialization values for global and static variables	FLASH
.econst	constants (e.g. const int k = 3;)	FLASH
.switch	tables for switch statements	FLASH
.pinit	tables for global constructors (C++)	FLASH

Uninitialized Sections

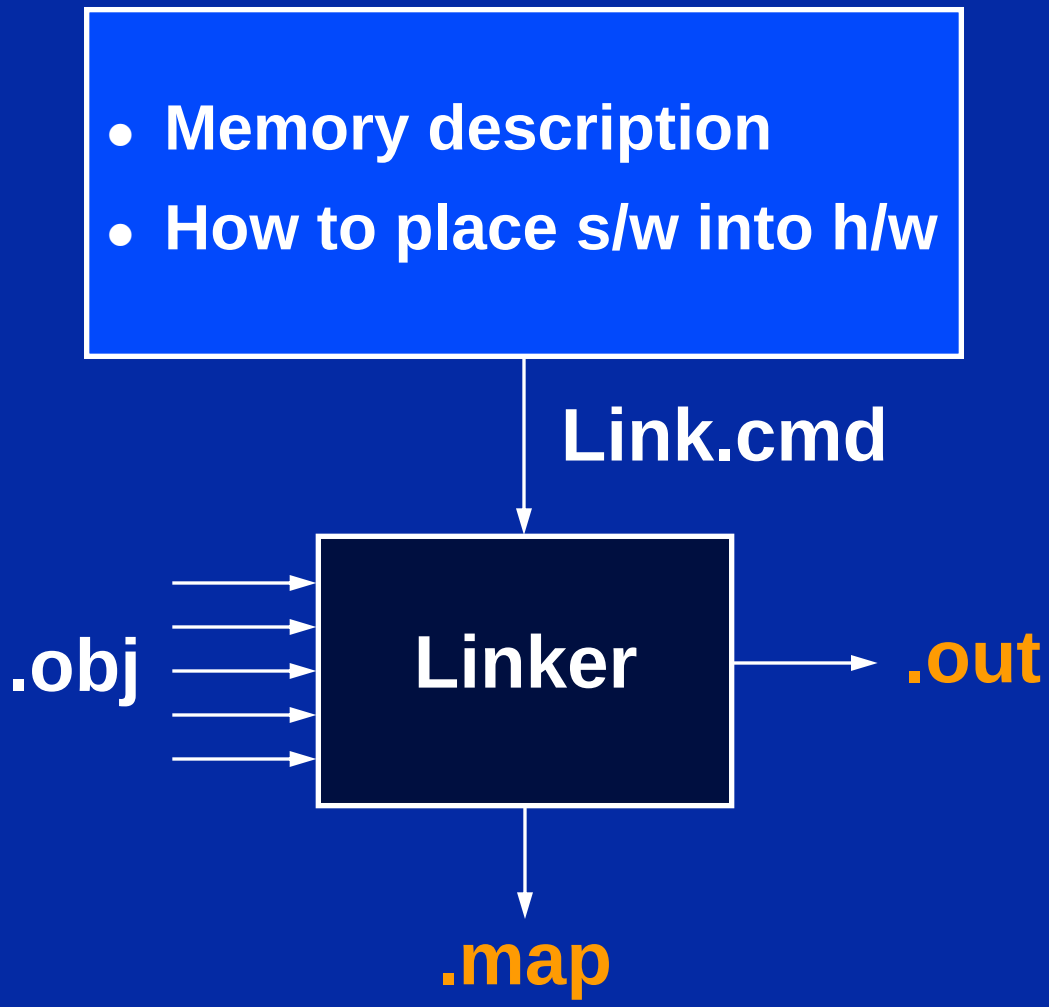
Name	Description	Link Location
.ebss	global and static variables	RAM
.stack	stack space	low 64Kw RAM
.esysmem	memory for far malloc functions	RAM

Note: During development initialized sections could be linked to RAM since the emulator can be used to load the RAM

Placing Sections in Memory



Linking


Memory description
How to place s/w into h/w
Link.cmd
.obj
Linker
.out
.map

Linker Command File

```
MEMORY
{
   PAGE 0:          /* Program Memory */
    FLASH:    origin = 0x3E8000, length = 0x10000

   PAGE 1:          /* Data Memory */
    M0SARAM:  origin = 0x000000, length = 0x400
    M1SARAM:  origin = 0x000400, length = 0x400
}
SECTIONS
{
    .text:>          FLASH       PAGE = 0
    .ebss:>          M0SARAM     PAGE = 1
    .cinit:>         FLASH       PAGE = 0
    .stack:>         M1SARAM     PAGE = 1
}
```

C2000 MCU 1-Day Workshop Outline

- **Workshop Introduction**
- **Architecture Overview**
- **Programming Development Environment**
  - *Lab: Linker command file*
- **Peripheral Register Header Files**
- **Reset, Interrupts and System Initialization**
  - *Lab: Watchdog and interrupts*
- **Control Peripherals**
  - *Lab: Generate and graph a PWM waveform*
- **Flash Programming**
  - *Lab: Run the code from flash memory*
- **The Next Step…**

Traditional Approach to C Coding

```
#define ADCCTL1       (volatile unsigned  int *)0x00007100
                                          ...
void main(void)
{
    *ADCCTL1 = 0x1234;                  //write entire register
    *ADCCTL1 |= 0x4000;                 //enable ADC module
}
```

Advantages

- Simple, fast and easy to type
- Variable names exactly match register names (easy to remember)

Disadvantages

- Requires individual masks to be generated to manipulate individual bits
- Cannot easily display bit fields in debugger window
- Will generate less efficient code in many cases

Structure Approach to C Coding

```
void main(void)
{
    AdcRegs.ADCCTL1.all = 0x1234;         //write entire register
    AdcRegs.ADCCTL1.bit.ADCENABLE = 1;    //enable ADC module
}
```

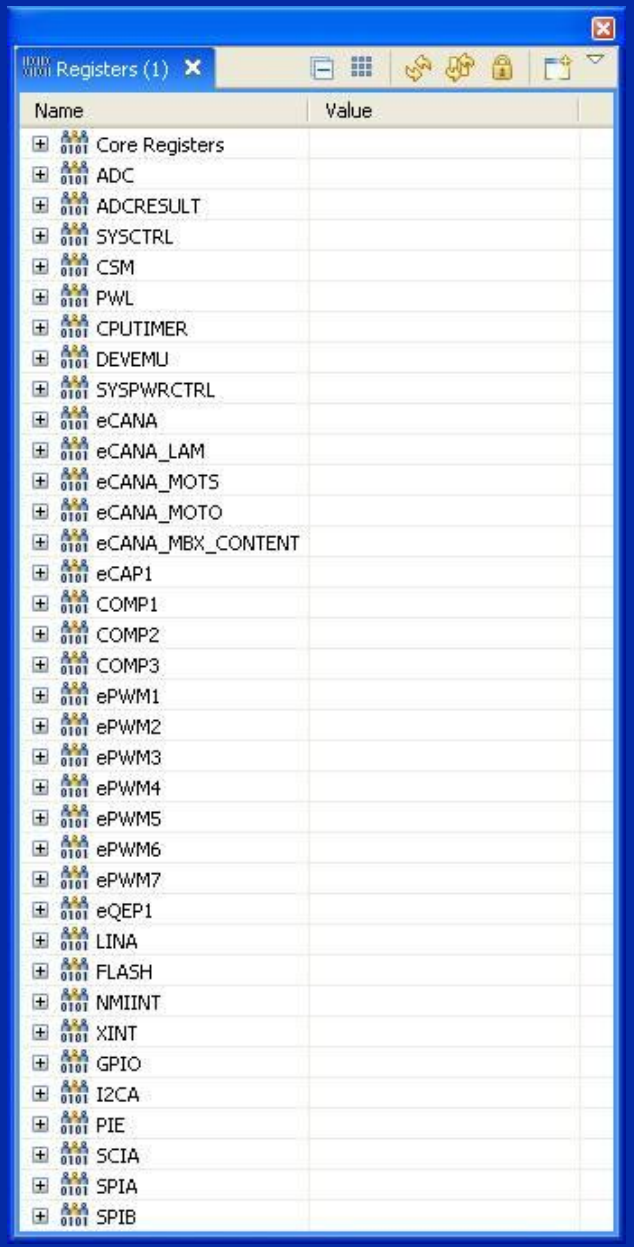
Advantages

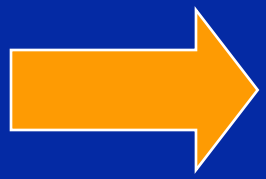
- Easy to manipulate individual bits
- Watch window is amazing! (next slide)
- Generates most efficient code (on C28x)

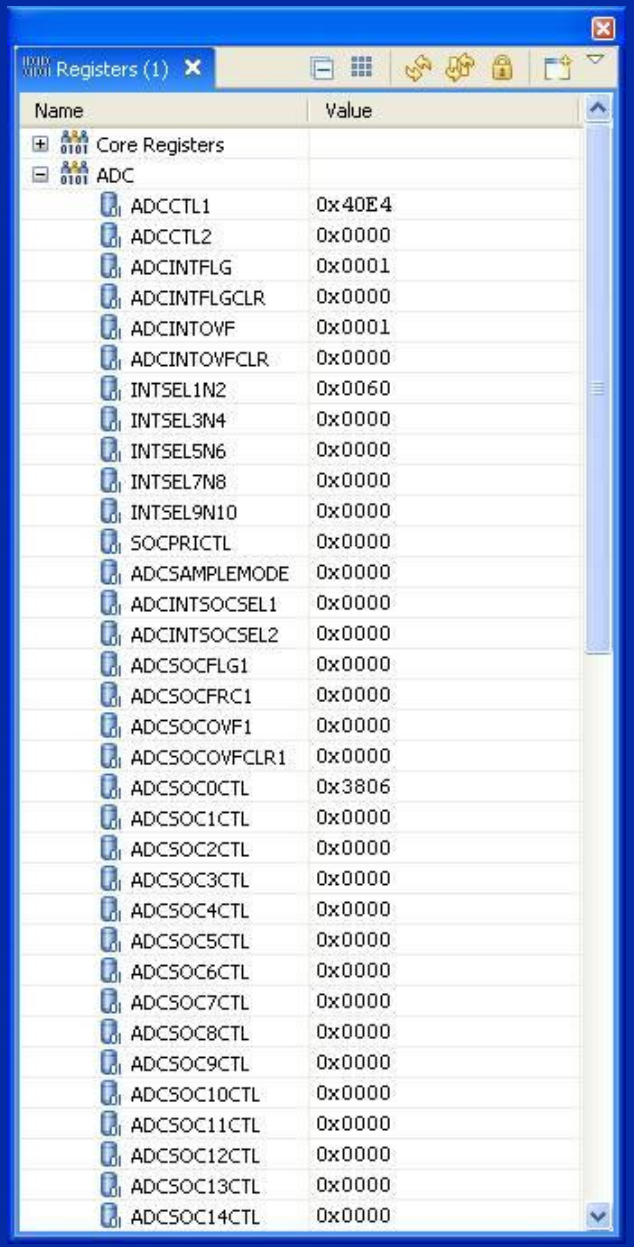
Disadvantages

- Can be difficult to remember the structure names (Editor Auto Complete feature to the rescue!)
- More to type (again, Editor Auto Complete feature to the rescue)

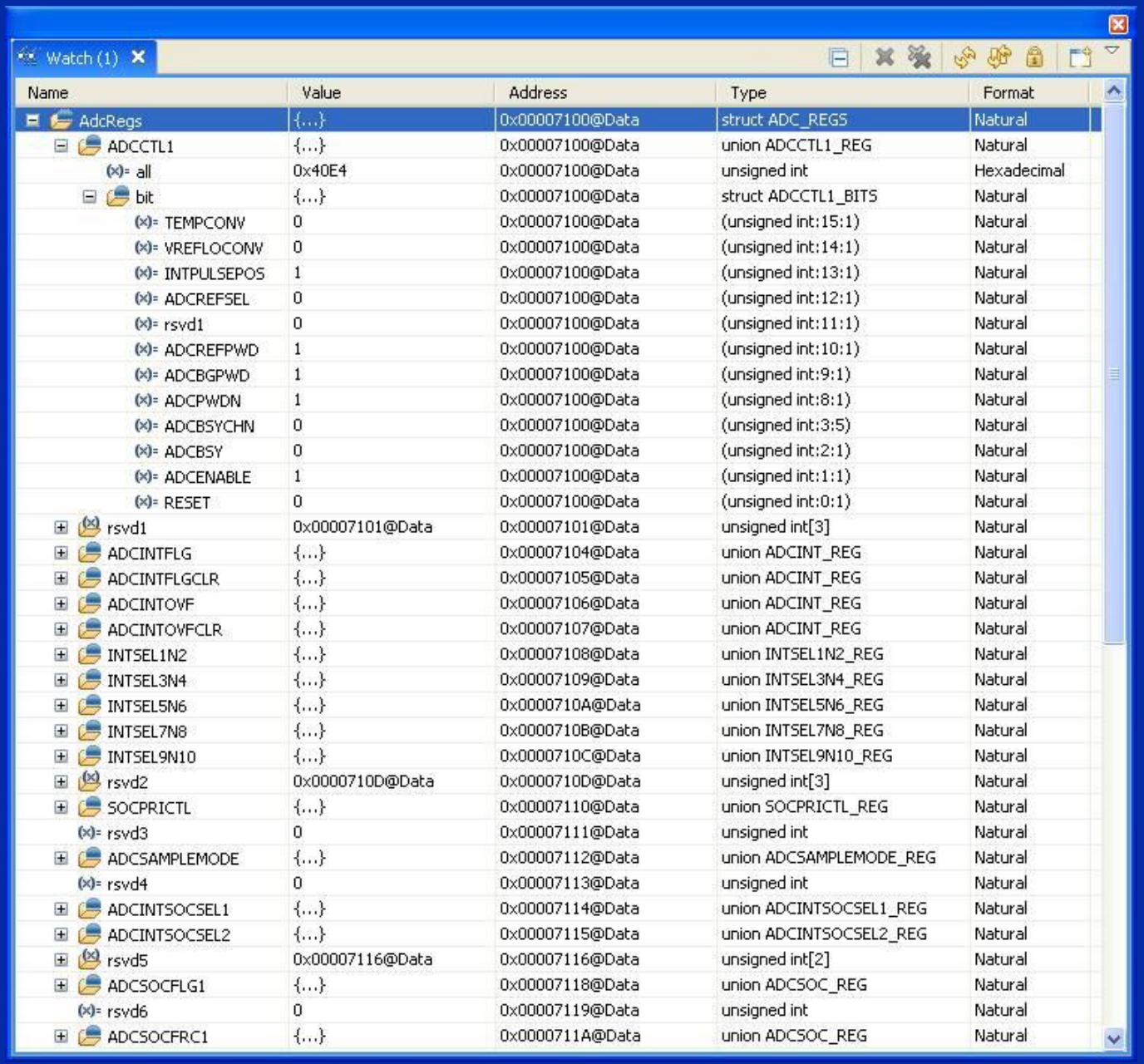
Built-in CCSv4 Register Window







CCSv4 Watch Window using Structures



Watch (1)

Name	Value	Address	Type	Format
AdcRegs	{...}	0x00007100@Data	struct ADC_REGS	Natural
ADCCTL1	{...}	0x00007100@Data	union ADCCTL1_REG	Natural
all	0x40E4	0x00007100@Data	unsigned int	Hexadecimal
bit	{...}	0x00007100@Data	struct ADCCTL1_BITS	Natural
TEMPCONV	0	0x00007100@Data	(unsigned int:15:1)	Natural
VREFLOCONV	0	0x00007100@Data	(unsigned int:14:1)	Natural
INTPULSEPOS	1	0x00007100@Data	(unsigned int:13:1)	Natural
ADCREFSEL	0	0x00007100@Data	(unsigned int:12:1)	Natural
rsvd1	0	0x00007100@Data	(unsigned int:11:1)	Natural
ADCREFPWD	1	0x00007100@Data	(unsigned int:10:1)	Natural
ADCBGPWD	1	0x00007100@Data	(unsigned int:9:1)	Natural
ADCPWDN	1	0x00007100@Data	(unsigned int:8:1)	Natural
ADCBSYCHN	0	0x00007100@Data	(unsigned int:3:5)	Natural
ADCBSY	0	0x00007100@Data	(unsigned int:2:1)	Natural
ADCENABLE	1	0x00007100@Data	(unsigned int:1:1)	Natural
RESET	0	0x00007100@Data	(unsigned int:0:1)	Natural
rsvd1	0x00007101@Data	0x00007101@Data	unsigned int[3]	Natural
ADCINTFLG	{...}	0x00007104@Data	union ADCINT_REG	Natural
ADCINTFLGCLR	{...}	0x00007105@Data	union ADCINT_REG	Natural
ADCINTOVF	{...}	0x00007106@Data	union ADCINT_REG	Natural
ADCINTOVFCLR	{...}	0x00007107@Data	union ADCINT_REG	Natural
INTSEL1N2	{...}	0x00007108@Data	union INTSEL1N2_REG	Natural
INTSEL3N4	{...}	0x00007109@Data	union INTSEL3N4_REG	Natural
INTSEL5N6	{...}	0x0000710A@Data	union INTSEL5N6_REG	Natural
INTSEL7N8	{...}	0x0000710B@Data	union INTSEL7N8_REG	Natural
INTSEL9N10	{...}	0x0000710C@Data	union INTSEL9N10_REG	Natural
rsvd2	0x0000710D@Data	0x0000710D@Data	unsigned int[3]	Natural
SOCPRICTL	{...}	0x00007110@Data	union SOCPRICTL_REG	Natural
rsvd3	0	0x00007111@Data	unsigned int	Natural
ADCSAMPLEMODE	{...}	0x00007112@Data	union ADCSAMPLEMODE_REG	Natural
rsvd4	0	0x00007113@Data	unsigned int	Natural
ADCINTSOCSEL1	{...}	0x00007114@Data	union ADCINTSOCSEL1_REG	Natural
ADCINTSOCSEL2	{...}	0x00007115@Data	union ADCINTSOCSEL2_REG	Natural
rsvd5	0x00007116@Data	0x00007116@Data	unsigned int[2]	Natural
ADCSOCFLG1	{...}	0x00007118@Data	union ADCSOC_REG	Natural
rsvd6	0	0x00007119@Data	unsigned int	Natural
ADCSOCFRC1	{...}	0x0000711A@Data	union ADCSOC_REG	Natural

Structure Naming Conventions

- The F2806x header files define:
  - All of the peripheral structures
  - All of the register names
  - All of the bit field names
  - All of the register addresses

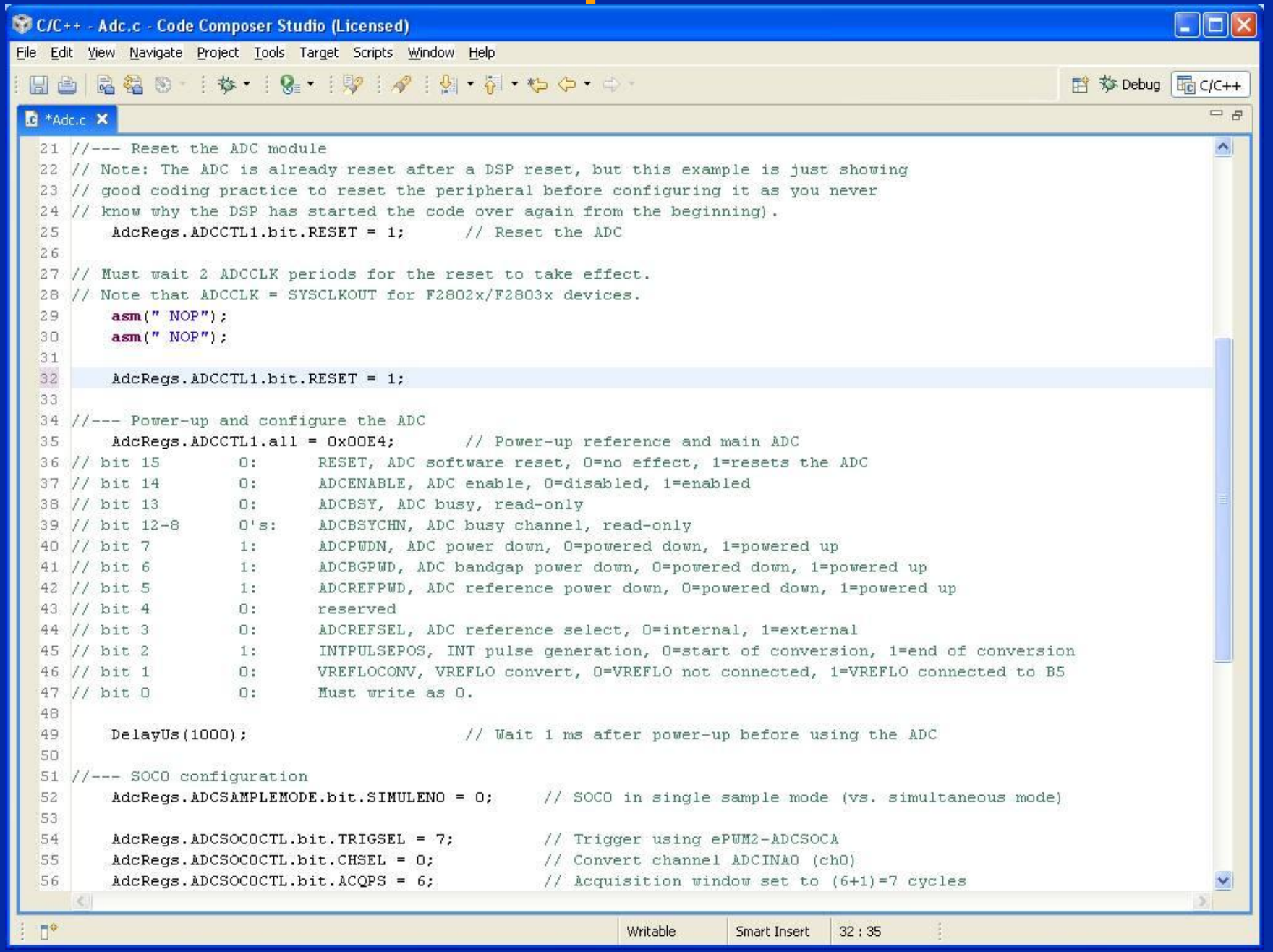
PeripheralName.RegisterName.all	// Access full 16 or 32-bit register
PeripheralName.RegisterName.half.LSW	// Access low 16-bits of 32-bit register
PeripheralName.RegisterName.half.MSW	// Access high 16-bits of 32-bit register
PeripheralName.RegisterName.bit.FieldName	// Access specified bit fields of register

Notes: [1] “PeripheralName” are assigned by TI and found in the F2806x header files. They are a combination of capital and small letters (i.e. CpuTimer0Regs).

[2] “RegisterName” are the same names as used in the data sheet. They are always in capital letters (i.e. TCR, TIM, TPR,..).

[3] “FieldName” are the same names as used in the data sheet. They are always in capital letters (i.e. POL, TOG, TSS,..).

Editor Auto Complete to the Rescue!



F2806x Header File Package

(http://www.ti.com, controlSUITE)

- Contains everything needed to use the structure approach
- Defines all peripheral register bits and register addresses
- Header file package includes:

\F2806x_headers\include	→ .h files
\F2806x_headers\cmd	→ linker .cmd files
\F2806x_common\gel	→ .gel files for CCS
\F2806x_examples	→ CCS4 examples
\doc	→ documentation

controlSUITE Header File Package located at `C:\TI\controlSUITE\device_support\`

Peripheral Structure .h files (1 of 2)

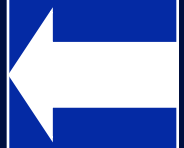
- Contain bits field structure definitions for each peripheral register

Your C-source file (e.g., Adc.c)

```
#include “F2806x_Device.h"

Void InitAdc(void)
{
   /* Reset the ADC module */
   AdcRegs.ADCCTL1.bit.RESET = 1;

   /* configure the ADC register */
   AdcRegs.ADCCTL1.all = 0x00E4;
};
```



F2806x_Adc.h

```
// ADC Individual Register Bit Definitions:
struct ADCCTL1_BITS {       // bits  description
   Uint16  TEMPCONV:1;      // 0 Temperature sensor connection
   Uint16  VREFLOCONV:1;    // 1 VSSA connection
   Uint16  INTPULSEPOS:1;   // 2 INT pulse generation control
   Uint16  ADCREFSEL:1;     // 3 Internal/external reference select
   Uint16  rsvd1:1;         // 4 reserved
   Uint16  ADCREFPWD:1;     // 5 Reference buffers powerdown
   Uint16  ADCBGPWD:1;      // 6 ADC bandgap powerdown
   Uint16  ADCPWDN:1;       // 7 ADC powerdown
   Uint16  ADCBSYCHN:5;     // 12:8 ADC busy on a channel
   Uint16  ADCBSY:1;        // 13 ADC busy signal
   Uint16  ADCENABLE:1;     // 14 ADC enable
   Uint16  RESET:1;         // 15 ADC master reset
};
// Allow access to the bit fields or entire register:
union ADCCTL1_REG {
   Uint16              all;
   struct ADCCTL1_BITS   bit;
};
// ADC External References & Function Declarations:
extern volatile struct ADC_REGS AdcRegs;
```

Peripheral Structure .h files (2 of 2)

- The header file package contains a .h file for each peripheral in the device

F2806x_Adc.h	F2806x_BootVars.h	F2806x_Cla.h
F2806x_Comp.h	F2806x_CpuTimers.h	F2806x_DevEmu.h
F2806x_Device.h	F2806x_Dma.h	F2806x_ECan.h
F2806x_ECap.h	F2806x_EPwm.h	F2806x_EQep.h
F2806x_Gpio.h	F2806x_I2c.h	F2806x_Mcbsp.h
F2806x_NmiIntrupt.h	F2806x_PieCtrl.h	F2806x_PieVect.h
F2806x_Sci.h	F2806x_Spi.h	F2806x_SysCtrl.h
F2806x_Usb.h	F2806x_XIntrupt.h	

- *F2806x_Device.h*
  - Main include file
  - Will include all other .h files
  - Include this file (*directly or indirectly*) in each source file:

```
#include   "F2806x_Device.h"
```

Global Variable Definitions File

F2806x_GlobalVariableDefs.c

- Declares a global instantiation of the structure for each peripheral
- Each structure is placed in its own section using a DATA_SECTION pragma to allow linking to the correct memory (see next slide)

F2806x_GlobalVariableDefs.c

```
#include "F2806x_Device.h"
…
#pragma DATA_SECTION(AdcRegs,"AdcRegsFile");
volatile struct ADC_REGS AdcRegs;
…
```

- Add this file to your CCS project:

F2806x_GlobalVariableDefs.c

Linker Command Files for the Structures

F2806x_nonBIOS.cmd and *F2806x_BIOS.cmd*

F2806x_GlobalVariableDefs.c

```
#include "F2806x_Device.h"
...
#pragma DATA_SECTION(AdcRegs,"AdcRegsFile");
volatile struct ADC_REGS AdcRegs;
...
```

F2806x_Headers_nonBIOS.cmd

```
MEMORY
{
  PAGE1:
    ...
    ADC:    origin=0x007100, length=0x000080
    ...
}
SECTIONS
{
   ...
   AdcRegsFile:    >  ADC        PAGE = 1
   ...
}
```

- Links each structure to the address of the peripheral using the structures named section
- non-BIOS and BIOS versions of the .cmd file
- Add one of these files to your CCS project:

  F2806x_nonBIOS.cmd

  or

  F2806x_BIOS.cmd

Peripheral Specific Examples

- Example projects for each peripheral
- Helpful to get you started

adc_soc
adc_temp_sensor
adc_temp_sensor_conv
cla_adc
cla_adc_fir
cla_adc_fir_flash
cpu_timer
dma_ram_to_ram
ecan_back2back
ecap_apwm
ecap_capture_pwm
epwm_blanking_window
epwm_dcevent_trip
epwm_dcevent_trip_comp
epwm_deadband
epwm_real-time_interrupts
epwm_timer_interrupts
epwm_trip_zone
epwm_up_aq
epwm_updown_aq
eqep_freqcal
eqep_pos_speed
external_interrupt
flash_f28069
fpu_hardware
fpu_software
gpio_setup
gpio_toggle
hrcap_capture_hrpwm
hrcap_capture_pwm
hrpwm
hrpwm_duty_sfo_v6
hrpwm_mult_ch_prdupdown_sfo_v6
hrpwm_prdup_sfo_v6
hrpwm_prdupdown_sfo_v6
hrpwm_slider
i2c_eeprom
lpm_haltwake
lpm_idlewake
lpm_standbywake
mcbsp_loopback
mcbsp_loopback_dma
mcbsp_loopback_interrupts
mcbsp_spi_loopback
osc_comp
sci_echoback
scia_loopback
scia_loopback_interrupts
spi_loopback
spi_loopback_interrupts
sw_prioritized_interrupts
timed_led_blink
usb_dev_bulk
usb_dev_chidcdc
usb_dev_keyboard
usb_dev_mouse
usb_dev_serial
usb_host_keyboard
usb_host_mouse
usb_host_msc
watchdog

Peripheral Register Header Files Summary

- Easier code development
- Easy to use
- Generates most efficient code
- Increases effectiveness of CCS watch window
- TI has already done all the work!
  - Use the correct header file package for your device:

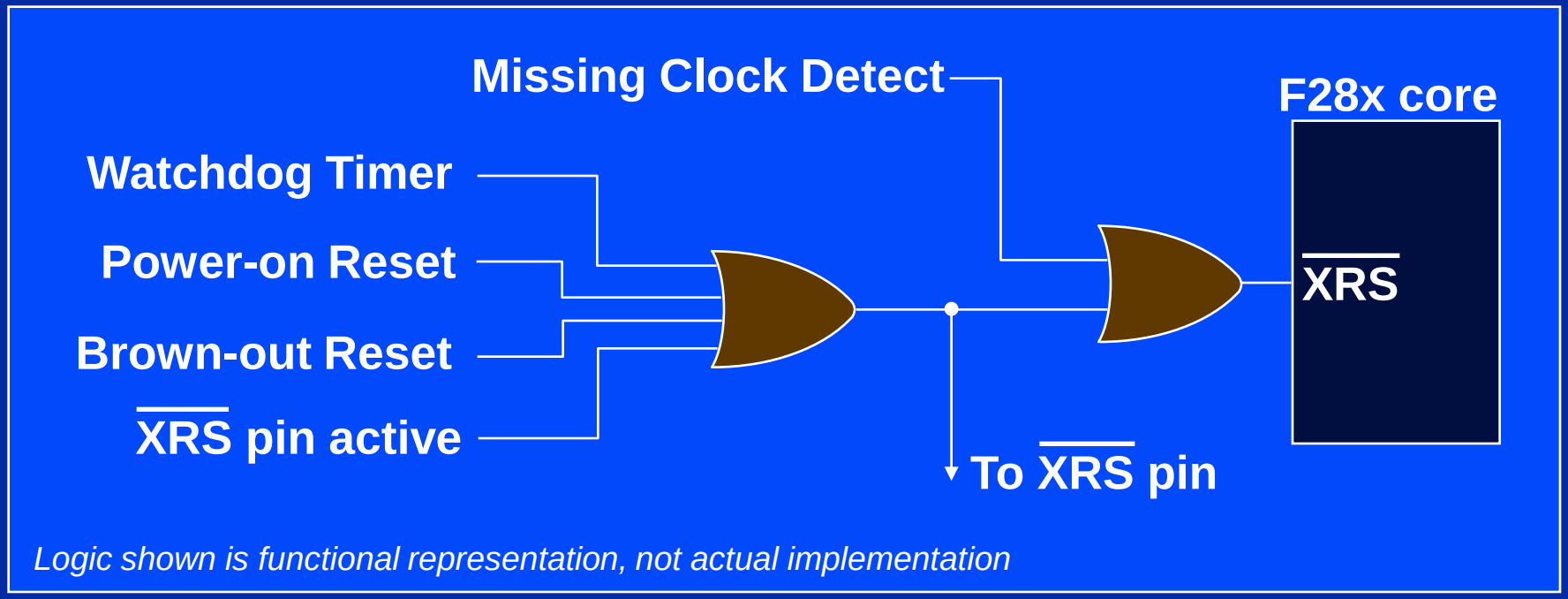
• F2806x	• F280x and F2801x
• F2803x	• F2804x
• F2802x	• F281x
• F2833x and F2823x	

Go to http://www.ti.com and enter “controlSUITE” in the keyword search box

C2000 MCU 1-Day Workshop Outline

- **Workshop Introduction**
- **Architecture Overview**
- **Programming Development Environment**
  - *Lab: Linker command file*
- **Peripheral Register Header Files**
- **Reset, Interrupts and System Initialization**
  - *Lab: Watchdog and interrupts*
- **Control Peripherals**
  - *Lab: Generate and graph a PWM waveform*
- **Flash Programming**
  - *Lab: Run the code from flash memory*
- **The Next Step…**

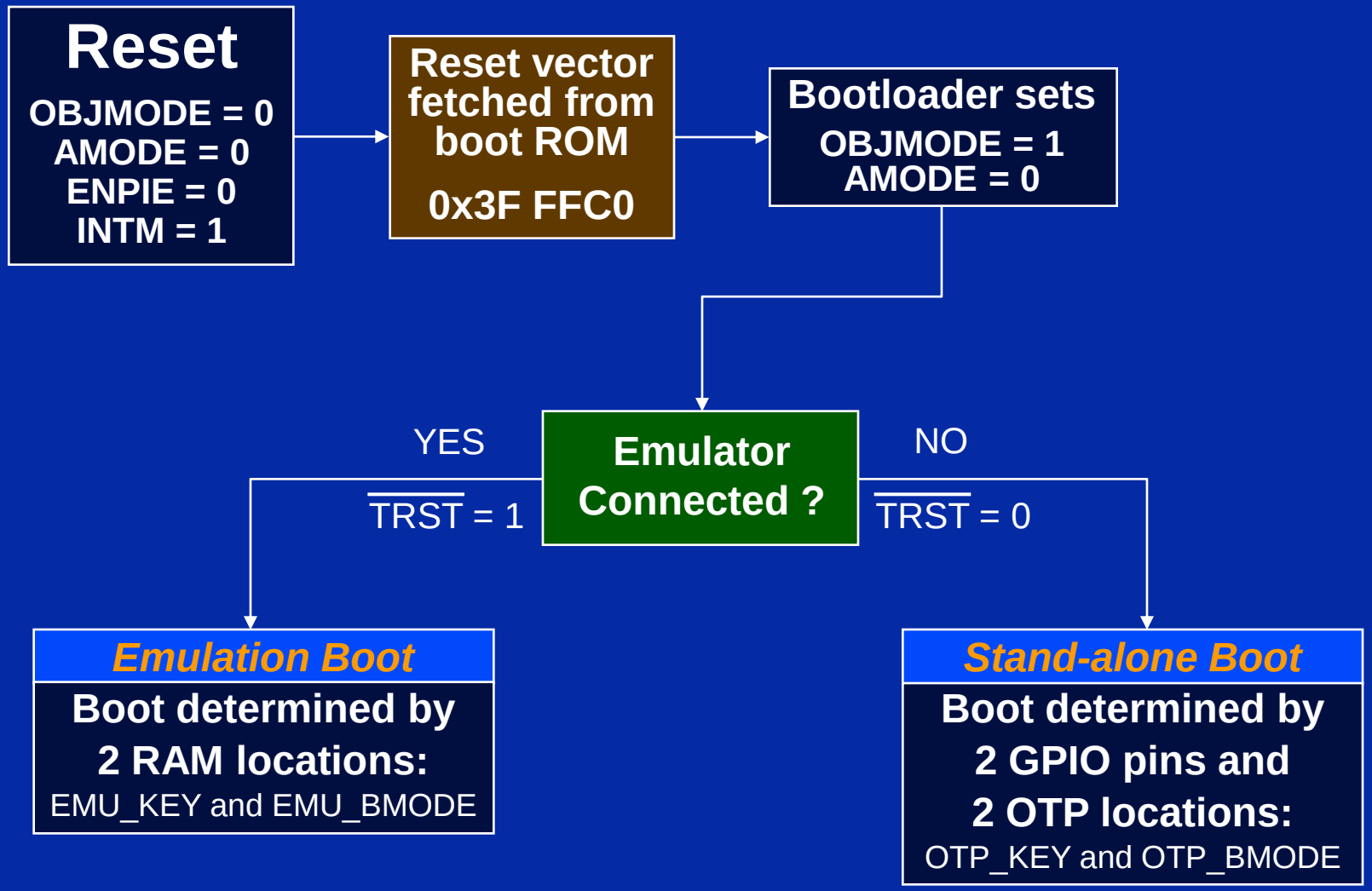
Reset Sources



- POR – *Power-on Reset* generates a device reset during power-up conditions
- BOR – *Brown-out Reset* generates a device reset if the power supply drops below specification for the device

Note: Devices support an on-chip voltage regulator (*VREG*) to generate the core voltage

Reset – Bootloader



TRST = JTAG Test Reset

EMU_KEY & EMU_BMODE located in PIE at 0x0D00 & 0x0D01, respectively
OTP_KEY & OTP_BMODE located in OTP at 0x3D7BFB & 0x3D7BFE, respectively

Emulation Boot Mode ($\overline{\text{TRST}}$ = 1)

Emulator Connected

Emulation Boot

Boot determined by 2 RAM locations:
EMU_KEY and EMU_BMODE

EMU_KEY = 0x55AA ?

NO → **Boot Mode: Wait**

YES

EMU_BMODE =	Boot Mode
0x0000	Parallel I/O
0x0001	SCI
0x0003	GetMode
0x0004	SPI
0x0005	I2C
0x0006	OTP
0x0007	CAN
0x000A	M0 SARAM
0x000B	FLASH
other	Wait

OTP_KEY = 0x005A ?

NO → **Boot Mode: FLASH**

YES

OTP_BMODE =	Boot Mode
0x0001	SCI
0x0004	SPI
0x0005	I2C
0x0006	OTP
0x0007	CAN
other	FLASH

If either EMU_KEY or EMU_BMODE are invalid, the "wait" boot mode is used. These values can then be modified using the debugger and a reset issued to restart the boot process

Stand-Alone Boot Mode ($\overline{TRST}$ = 0)

Emulator Not Connected

Stand-alone Boot

Boot determined by 2 GPIO pins and 2 OTP locations:
OTP_KEY and OTP_BMODE

Note that the boot behavior for unprogrammed OTP is the "FLASH" boot mode

GPIO 37	GPIO 34	Boot Mode
0	0	Parallel I/O
0	1	SCI
1	0	Wait
1	1	GetMode

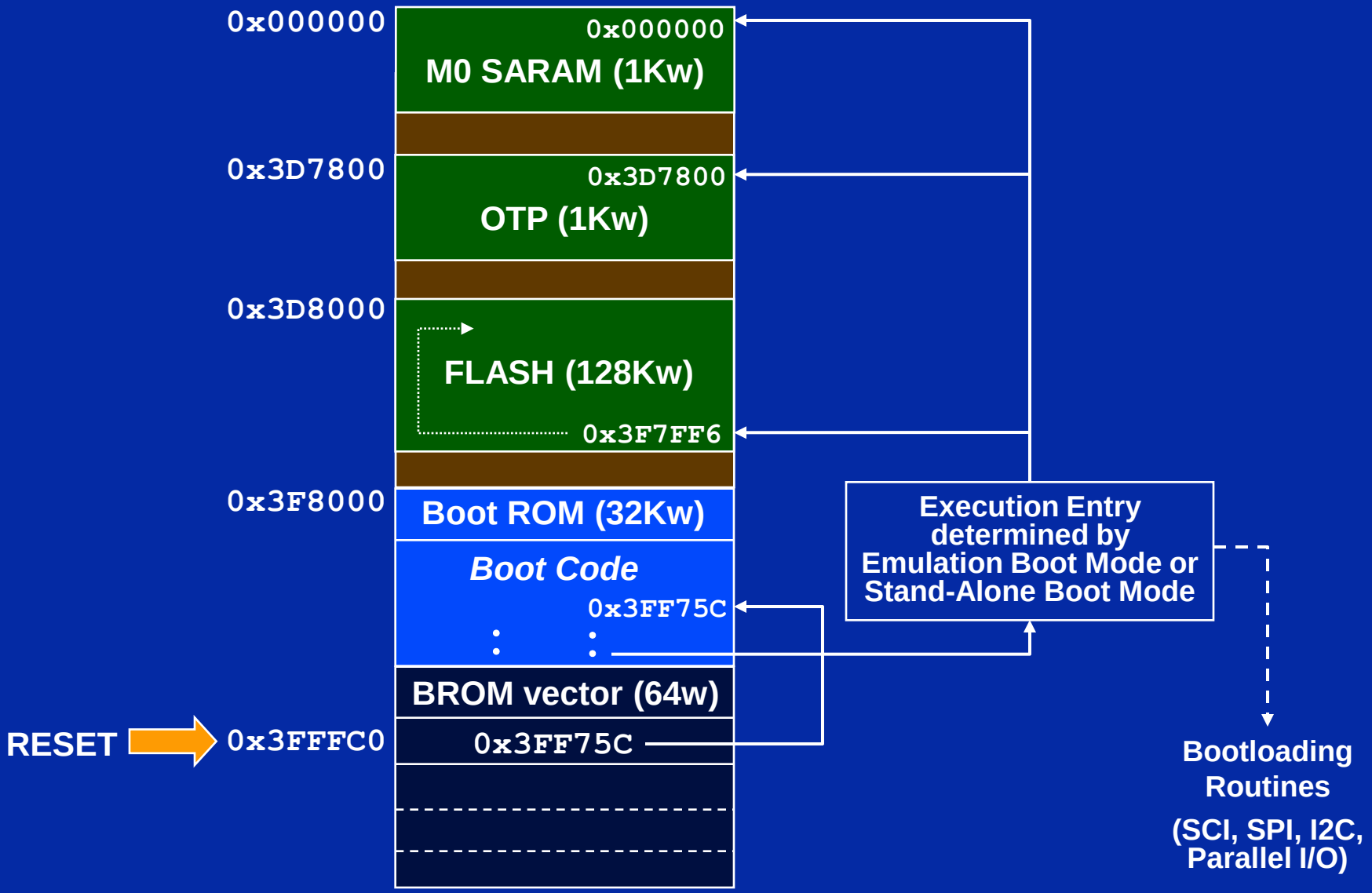
OTP_KEY = 0x005A ?

NO → **Boot Mode: FLASH**

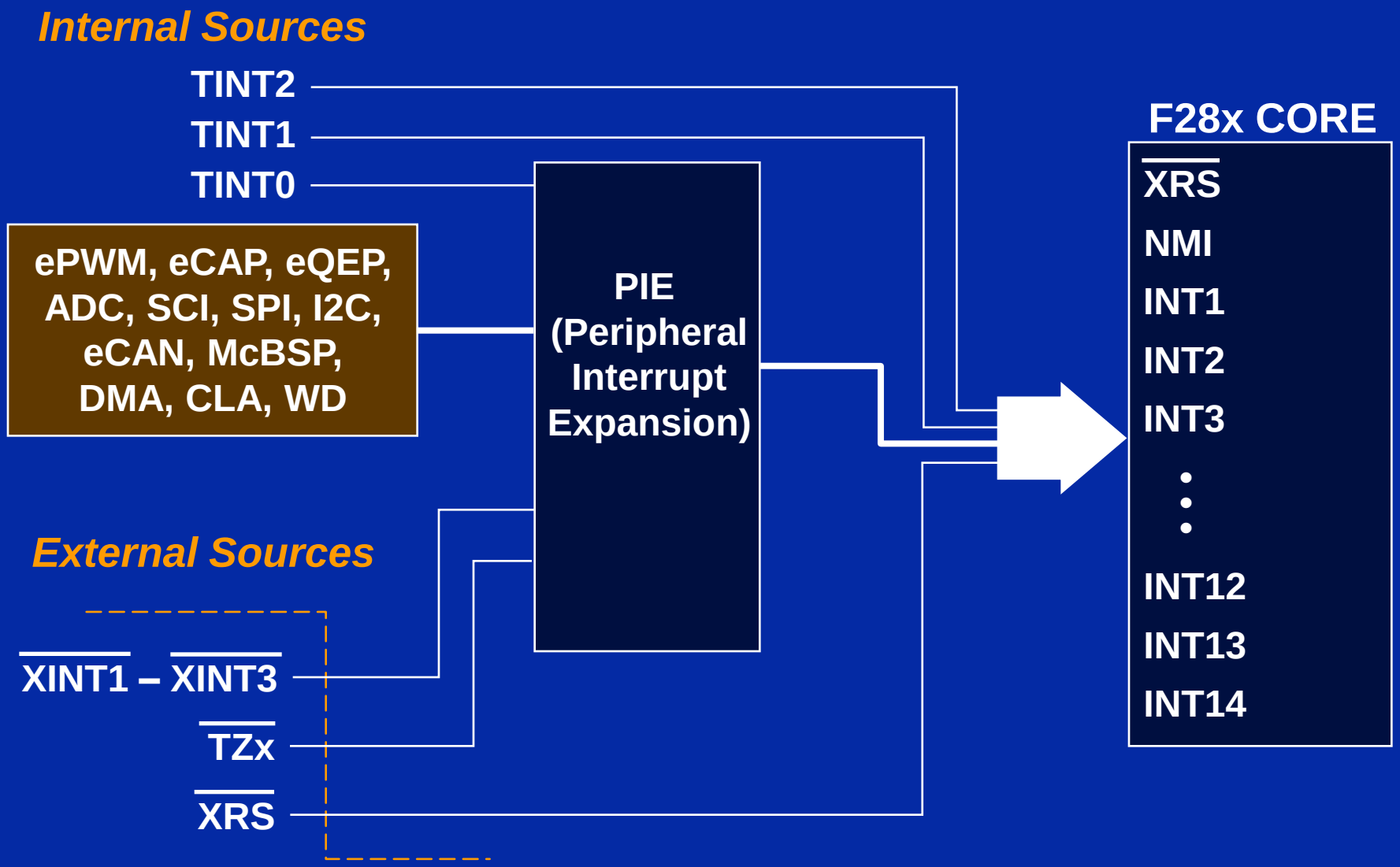
YES

OTP_BMODE =	Boot Mode
0x0001	SCI
0x0004	SPI
0x0005	I2C
0x0006	OTP
0x0007	CAN
other	FLASH

Reset Code Flow - Summary

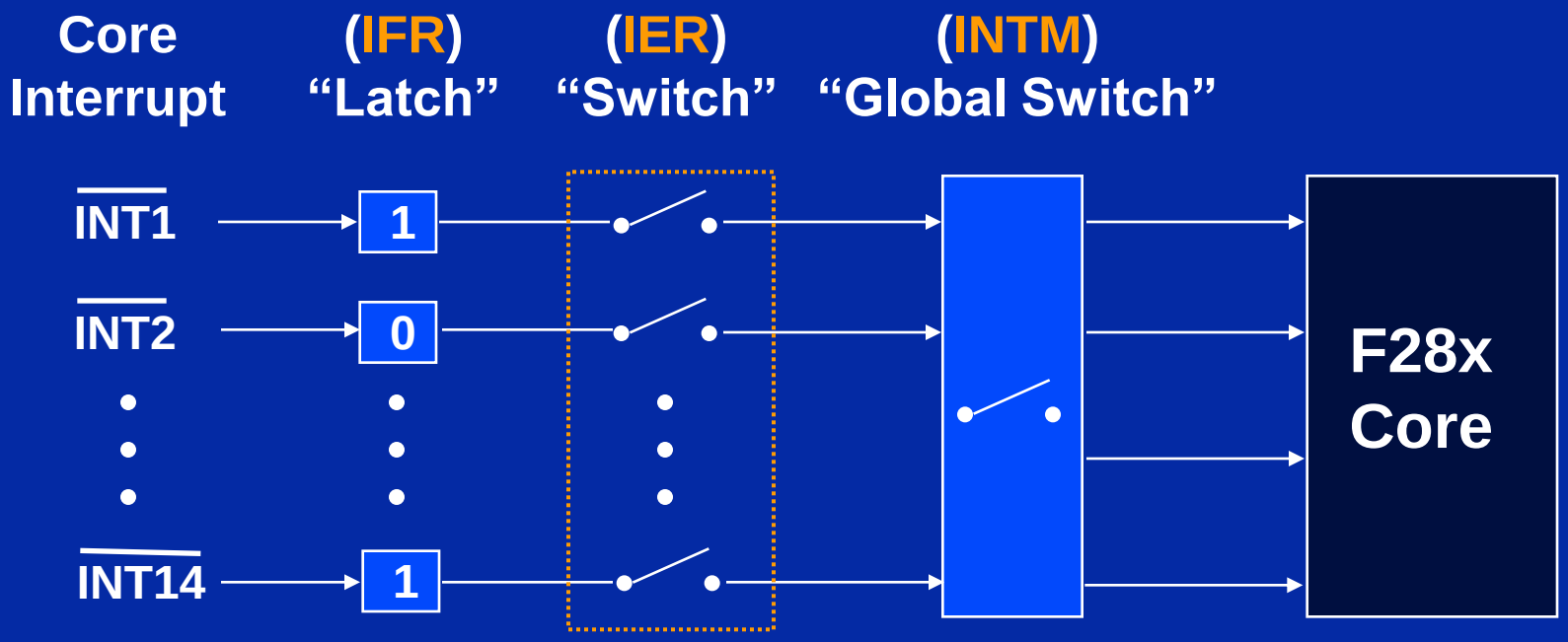


Interrupt Sources



Maskable Interrupt Processing

Conceptual Core Overview



- A valid signal on a specific interrupt line causes the latch to display a "1" in the appropriate bit
- If the individual and global switches are turned "on" the interrupt reaches the core

Core Interrupt Registers

Interrupt Flag Register (IFR)

(pending = 1 / absent = 0)

15	14	13	12	11	10	9	8
RTOSINT	DLOGINT	INT14	INT13	INT12	INT11	INT10	INT9
INT8	INT7	INT6	INT5	INT4	INT3	INT2	INT1
7	6	5	4	3	2	1	0

Interrupt Enable Register (IER)

(enable = 1 / disable = 0)

15	14	13	12	11	10	9	8
RTOSINT	DLOGINT	INT14	INT13	INT12	INT11	INT10	INT9
INT8	INT7	INT6	INT5	INT4	INT3	INT2	INT1
7	6	5	4	3	2	1	0

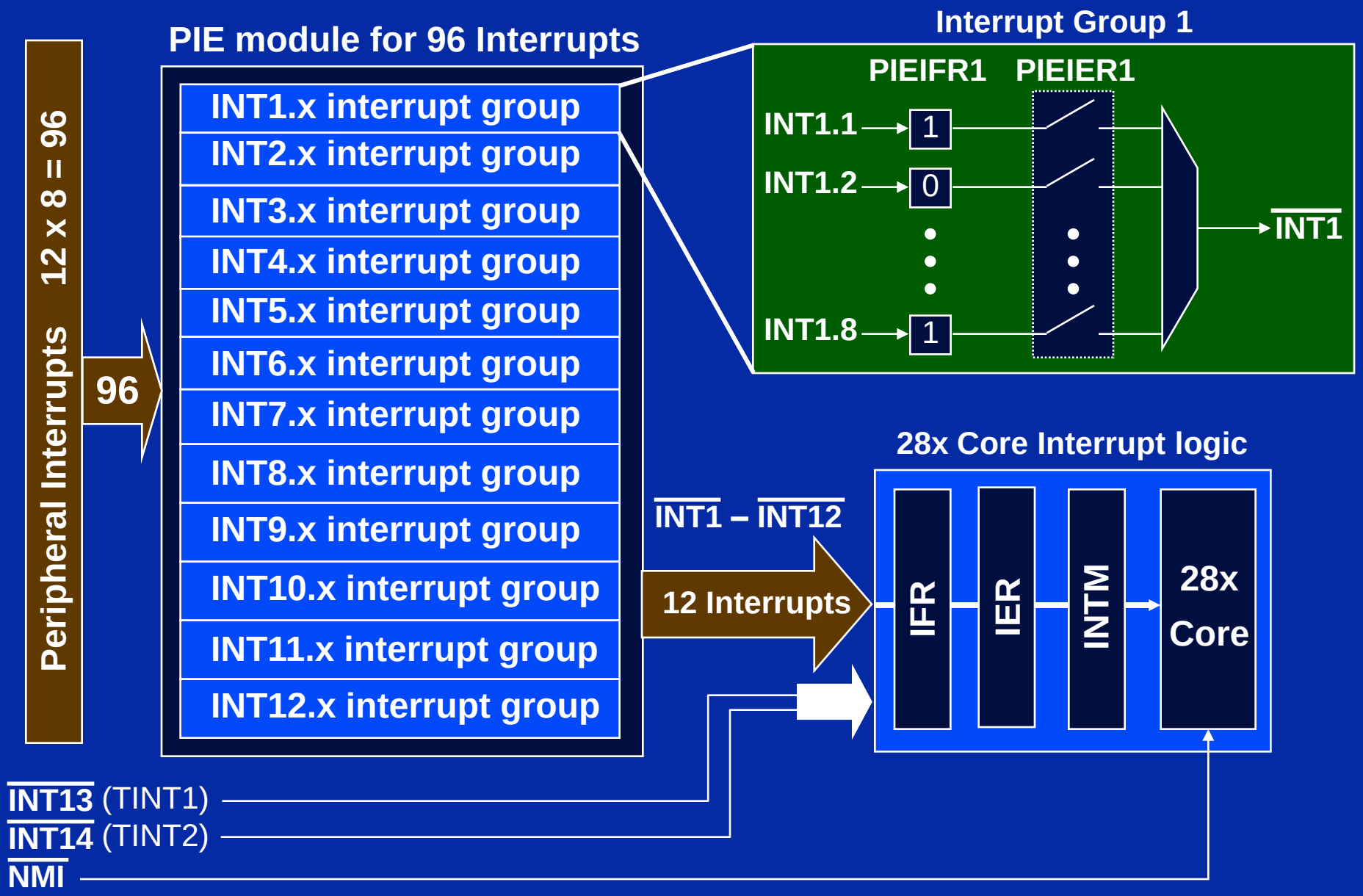
Interrupt Global Mask Bit (INTM)

		Bit 0
ST1		INTM

(enable = 0 / disable = 1)

```
/*** Interrupt Enable Register ***/
extern cregister volatile unsigned int IER;
      IER  |= 0x0008;              //enable INT4 in IER
      IER &= 0xFFF7;              //disable INT4 in IER
/*** Global Interrupts ***/
      asm(“  CLRC  INTM”);      //enable global interrupts
      asm(“  SETC  INTM”);      //disable global interrupts
```

Peripheral Interrupt Expansion - PIE



F2806x PIE Interrupt Assignment Table

	INTx.8	INTx.7	INTx.6	INTx.5	INTx.4	INTx.3	INTx.2	INTx.1
INT1	WAKEINT	TINT0	ADCINT9	XINT2	XINT1		ADCINT2	ADCINT1
INT2	EPWM8_TZINT	EPWM7_TZINT	EPWM6_TZINT	EPWM5_TZINT	EPWM4_TZINT	EPWM3_TZINT	EPWM2_TZINT	EPWM1_TZINT
INT3	EPWM8_INT	EPWM7_INT	EPWM6_INT	EPWM5_INT	EPWM4_INT	EPWM3_INT	EPWM2_INT	EPWM1_INT
INT4	HRCAP2_INT	HRCAP1_INT				ECAP3_INT	ECAP2_INT	ECAP1_INT
INT5				HRCAP4_INT	HRCAP3_INT		EQEP2_INT	EQEP1_INT
INT6			MXINTA	MRINTA	SPITXINTB	SPIRXINTB	SPITXINTA	SPIRXINTA
INT7			DINTCH6	DINTCH5	DINTCH4	DINTCH3	DINTCH2	DINTCH1
INT8							I2CINT2A	I2CINT1A
INT9			ECAN1_INTA	ECAN0_INTA	SCITXINTB	SCIRXINTB	SCITXINTA	SCIRXINTA
INT10	ADCINT8	ADCINT7	ADCINT6	ADCINT5	ADCINT4	ADCINT3	ADCINT2	ADCINT1
INT11	CLA1_INT8	CLA1_INT7	CLA1_INT6	CLA1_INT5	CLA1_INT4	CLA1_INT3	CLA1_INT2	CLA1_INT1
INT12	LUF	LVF						XINT3

PIE Registers

PIEIFRx register (x = 1 to 12)

15 - 8	7	6	5	4	3	2	1	0
reserved	INTx.8	INTx.7	INTx.6	INTx.5	INTx.4	INTx.3	INTx.2	INTx.1

PIEIERx register (x = 1 to 12)

15 - 8	7	6	5	4	3	2	1	0
reserved	INTx.8	INTx.7	INTx.6	INTx.5	INTx.4	INTx.3	INTx.2	INTx.1

PIE Interrupt Acknowledge Register (PIEACK)

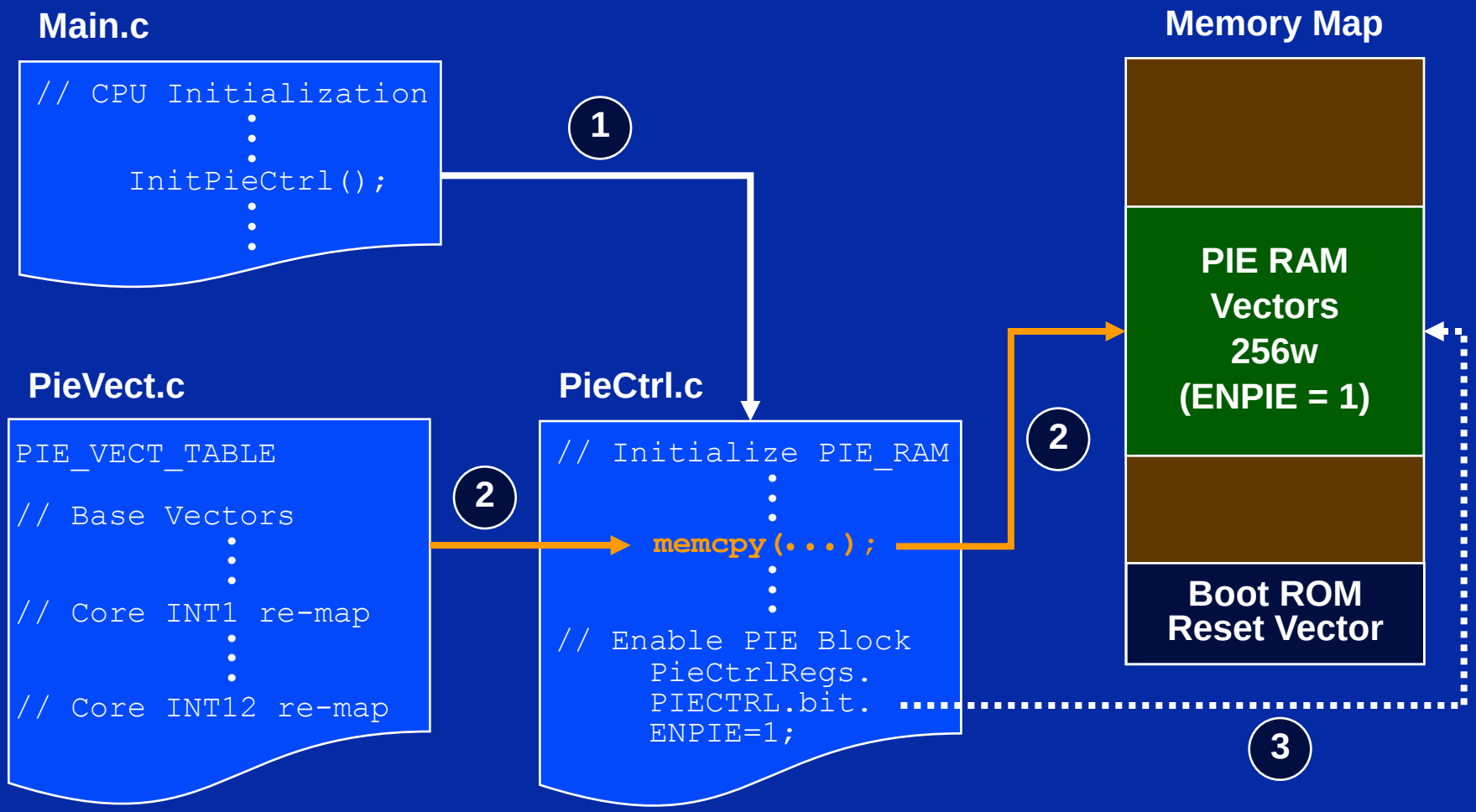
15 - 12	11	10	9	8	7	6	5	4	3	2	1	0
reserved	PIEACKx											

PIECTRL register

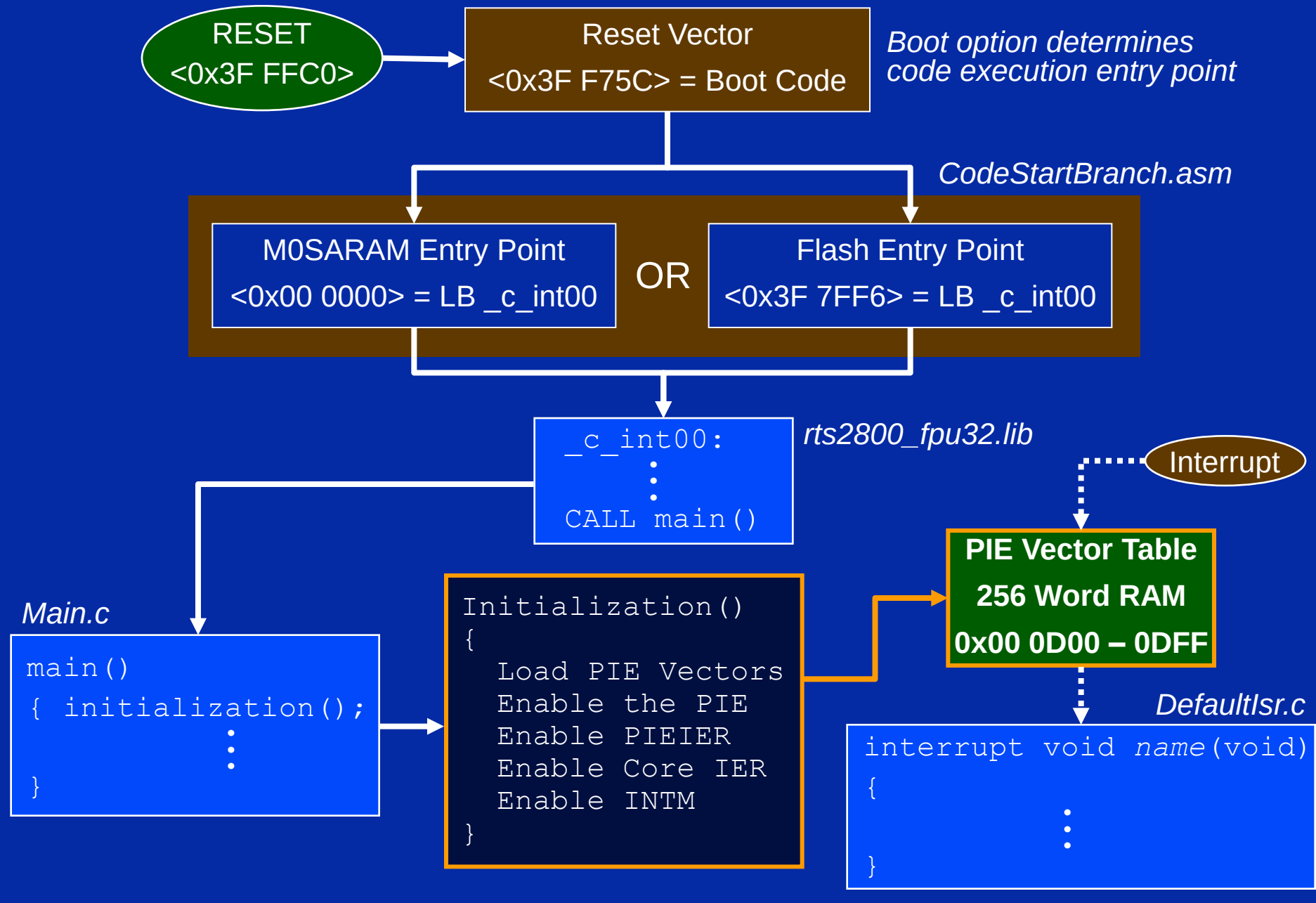
15 - 1	0
PIEVECT	ENPIE

```
#include “F2806x_Device.h”
    PieCtrlRegs.PIEIFR1.bit.INTx4 = 1;    //manually set IFR for XINT1 in PIE group 1
    PieCtrlRegs.PIEIER3.bit.INTx2 = 1;    //enable EPWM2_INT in PIE group 3
    PieCtrlRegs.PIEACK.all = 0x0004;      //acknowledge the PIE group 3
    PieCtrlRegs.PIECTRL.bit.ENPIE = 1;  //enable the PIE
```

PIE Block Initialization

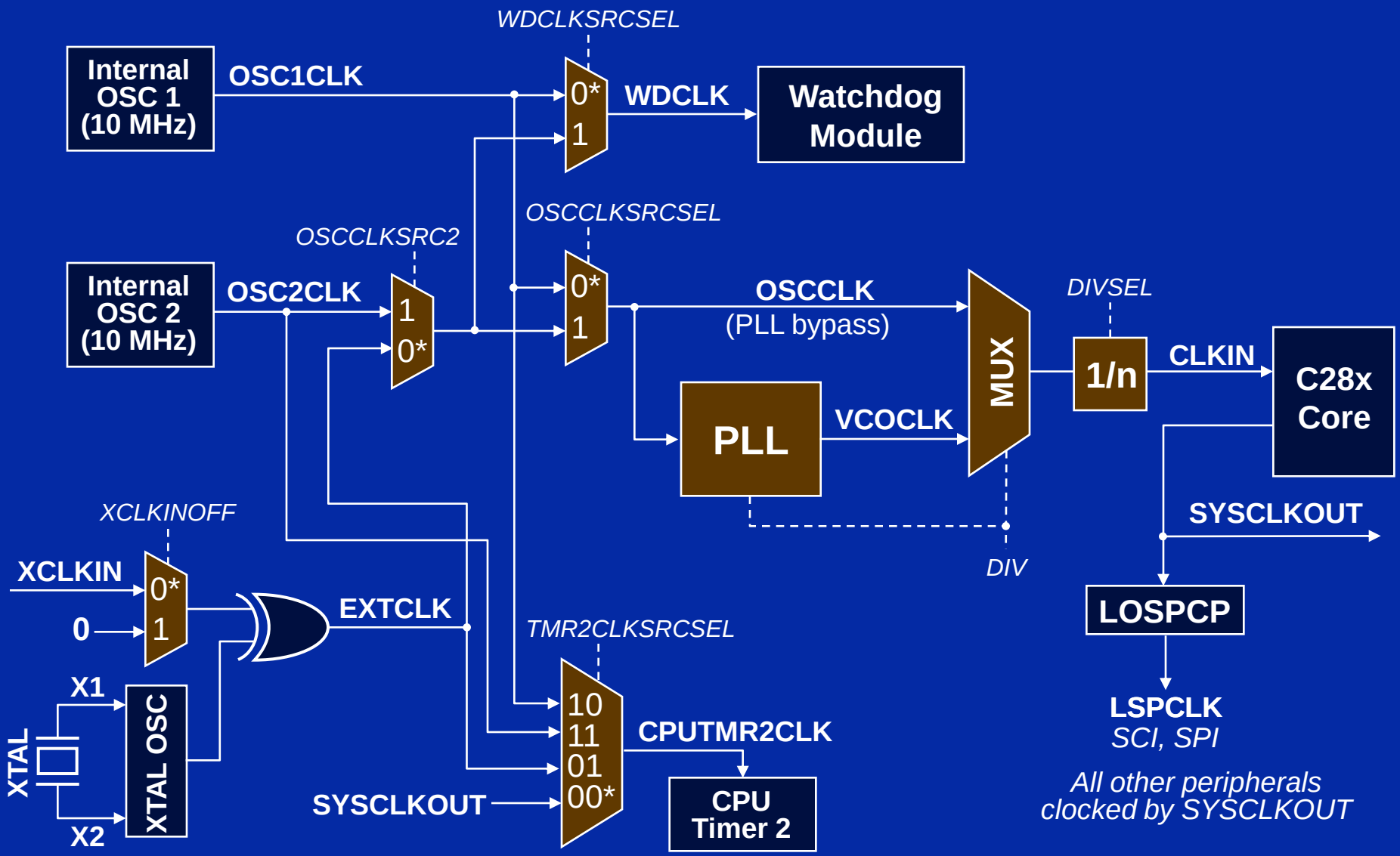


PIE Initialization Code Flow - Summary



F2806x Oscillator / PLL Clock Module

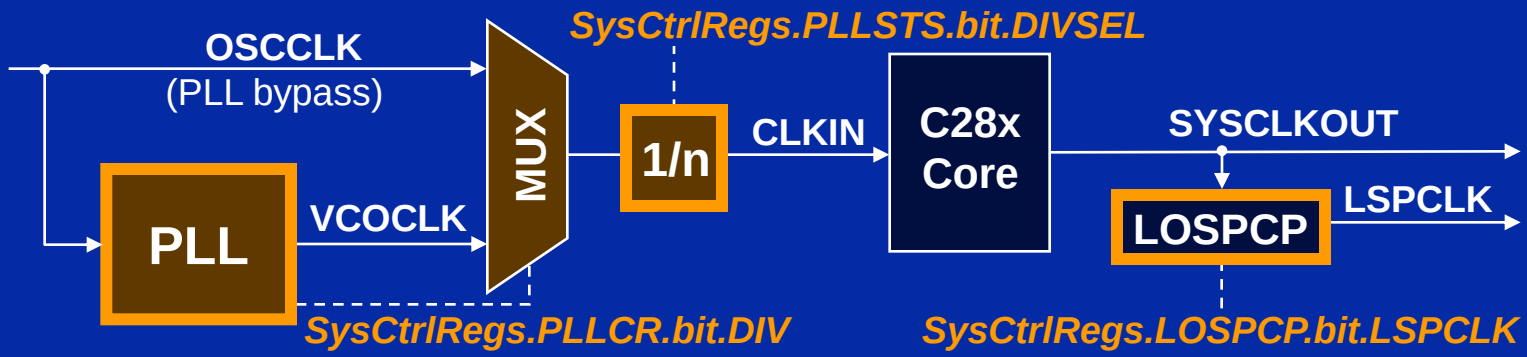
(lab file: SysCtrl.c)



* = default

F2806x PLL and LOSPCP

(lab file: SysCtrl.c)



DIV	CLKIN
0 0 0 0 0	OSCCLK / n * (PLL bypass)
0 0 0 0 1	OSCCLK x 1 / n
0 0 0 1 0	OSCCLK x 2 / n
0 0 0 1 1	OSCCLK x 3 / n
0 0 1 0 0	OSCCLK x 4 / n
0 0 1 0 1	OSCCLK x 5 / n
0 0 1 1 0	OSCCLK x 6 / n
0 0 1 1 1	OSCCLK x 7 / n
0 1 0 0 0	OSCCLK x 8 / n
0 1 0 0 1	OSCCLK x 9 / n
0 1 0 1 0	OSCCLK x 10 / n
0 1 0 1 1	OSCCLK x 11 / n
0 1 1 0 0	OSCCLK x 12 / n
0 1 1 0 1	OSCCLK x 13 / n
0 1 1 1 0	OSCCLK x 14 / n
0 1 1 1 1	OSCCLK x 15 / n
1 0 0 0 0	OSCCLK x 16 / n
1 x x x 1	reserved

DIVSEL	n
0x	/4 *
10	/2
11	/1

*** default**

Note: /1 mode can only be used when PLL is bypassed

LSPCLK	Peripheral Clk Freq
0 0 0	SYSCLKOUT / 1
0 0 1	SYSCLKOUT / 2
0 1 0	SYSCLKOUT / 4 *
0 1 1	SYSCLKOUT / 6
1 0 0	SYSCLKOUT / 8
1 0 1	SYSCLKOUT / 10
1 1 0	SYSCLKOUT / 12
1 1 1	SYSCLKOUT / 14

LSBs in reg. – others reserved

Input Clock Fail Detect Circuitry

PLL will issue a "limp mode" clock (1-4 MHz) if input clock is removed after PLL has locked.

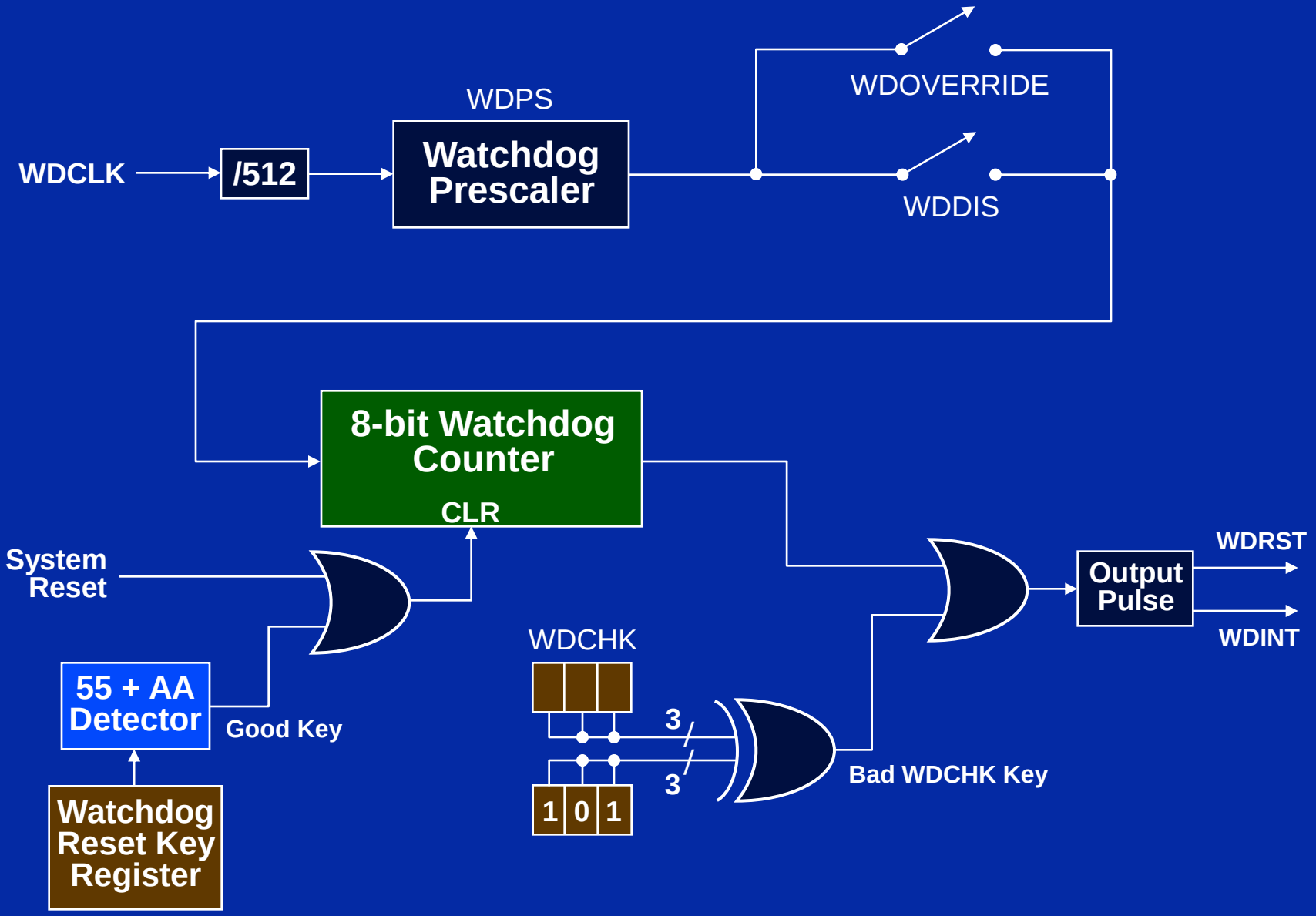
An internal device reset will also be issued (XRSn pin not driven).

Watchdog Timer

- Resets the C28x if the CPU crashes
  - Watchdog counter runs independent of CPU
  - If counter overflows, a reset or interrupt is triggered (user selectable)
  - CPU must write correct data key sequence to reset the counter before overflow
- Watchdog must be serviced or disabled within 131,072 WDCLK cycles after reset
- This translates to 13.11 ms with a 10 MHz WDCLK

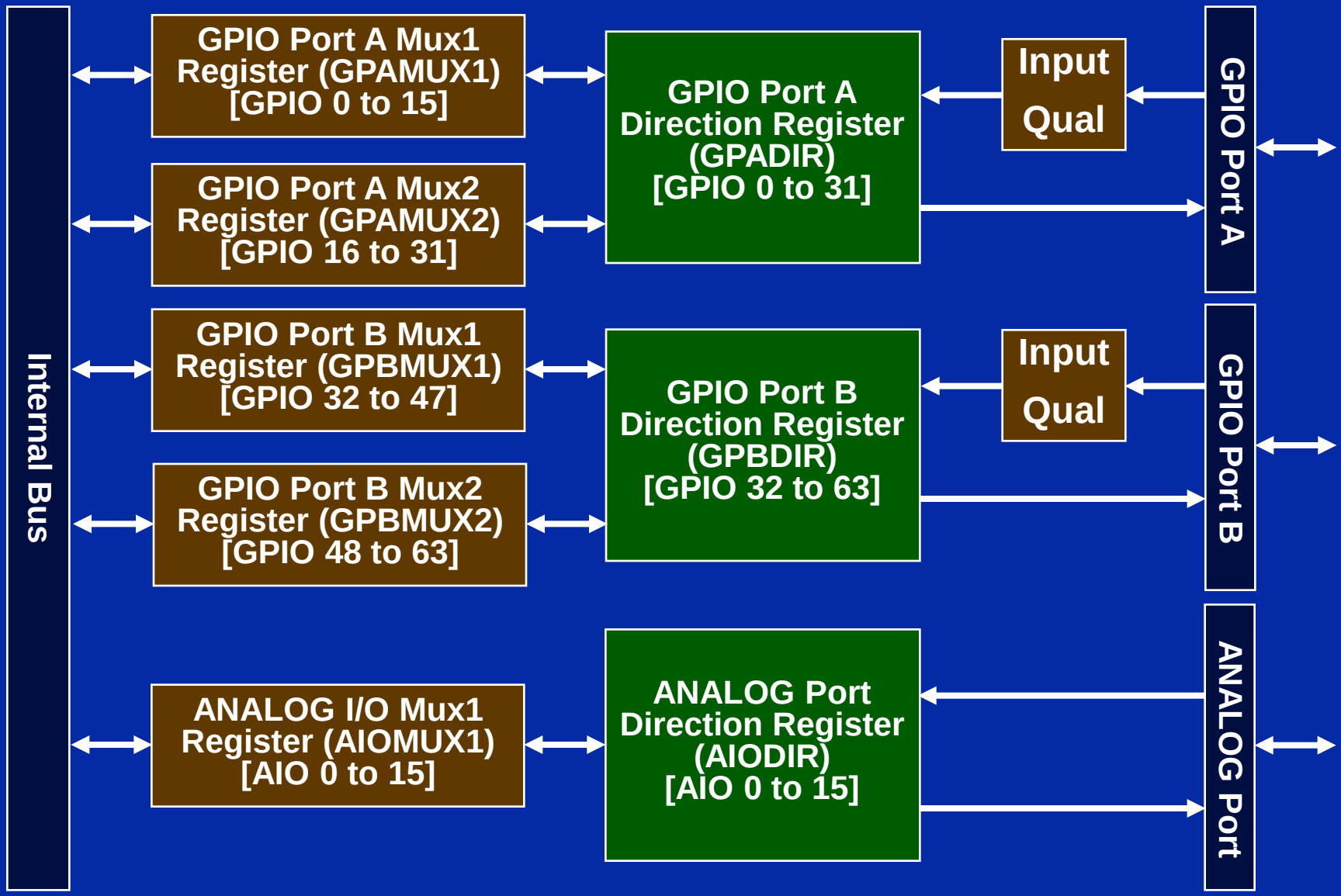
Watchdog Timer Module

(lab file: Watchdog.c)



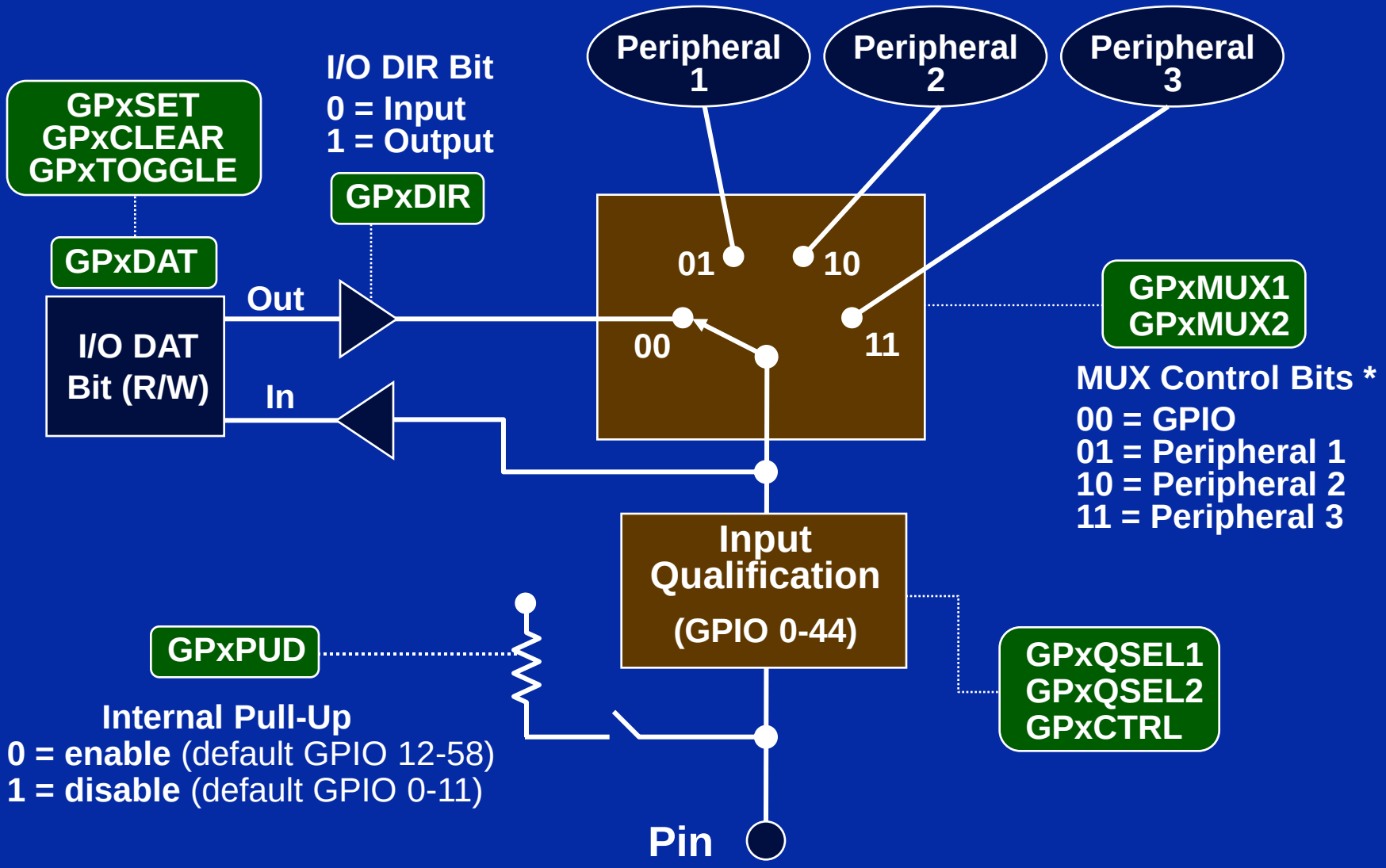
F2806x GPIO Grouping Overview

(lab file: Gpio.c)



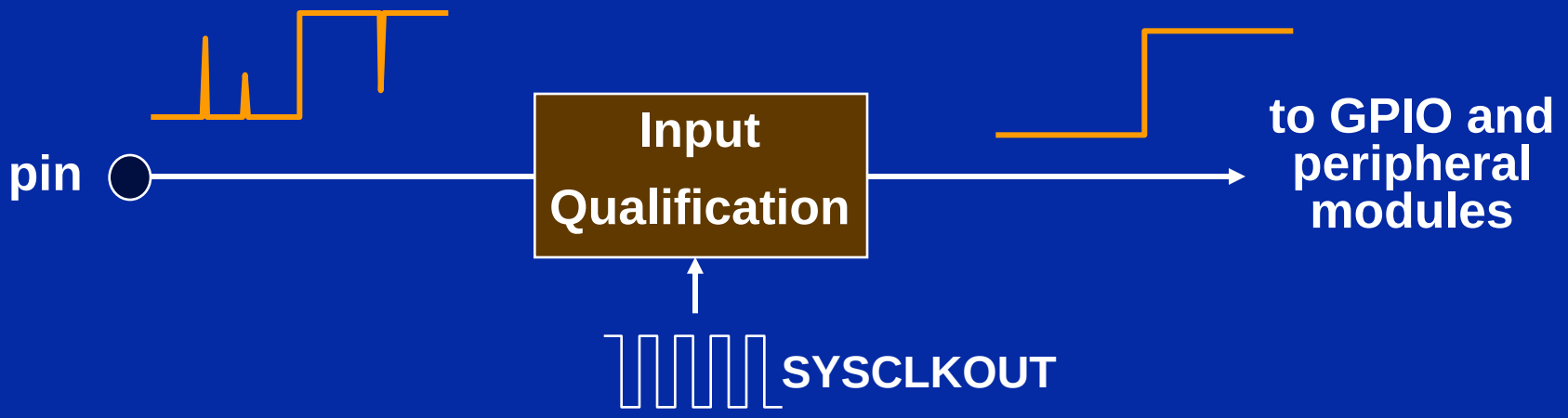
F2806x GPIO Pin Block Diagram

(lab file: Gpio.c)

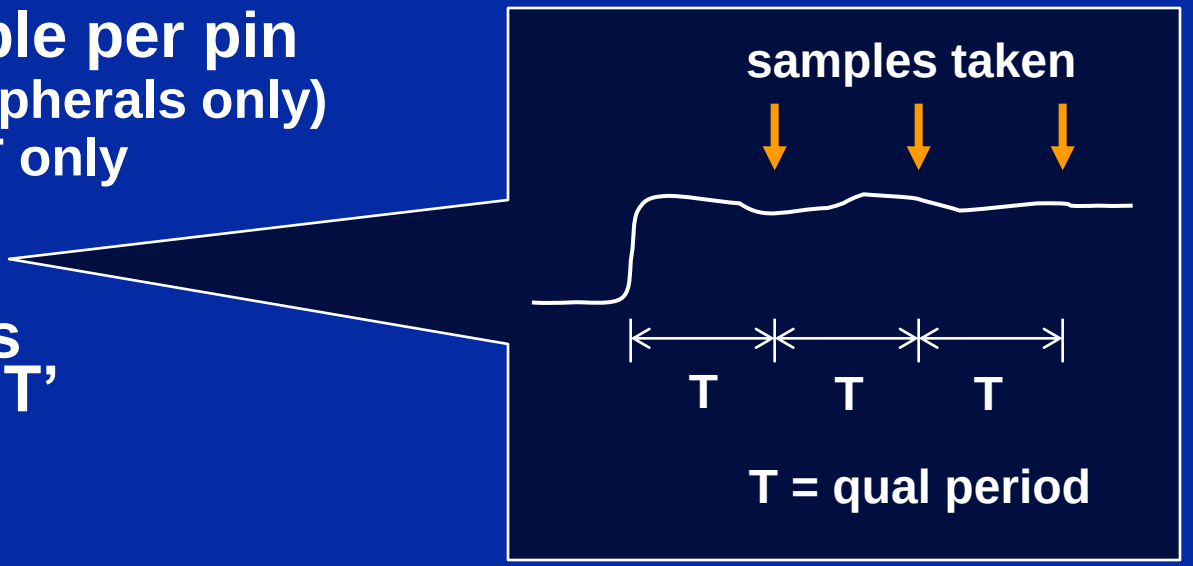


** See device datasheet for pin function selection matrices*

F2806x GPIO Input Qualification



- Qualification available on ports A & B only
- Individually selectable per pin
  - no qualification (peripherals only)
  - sync to SYSCLKOUT only
  - qualify 3 samples
  - qualify 6 samples
- AIO pins are fixed as ‘sync to SYSCLKOUT’



Lab 1: System Initialization

- LAB1 files have been provided
- LAB1 consists of two parts:

  Part 1

  - Test behavior of watchdog when disabled and enabled

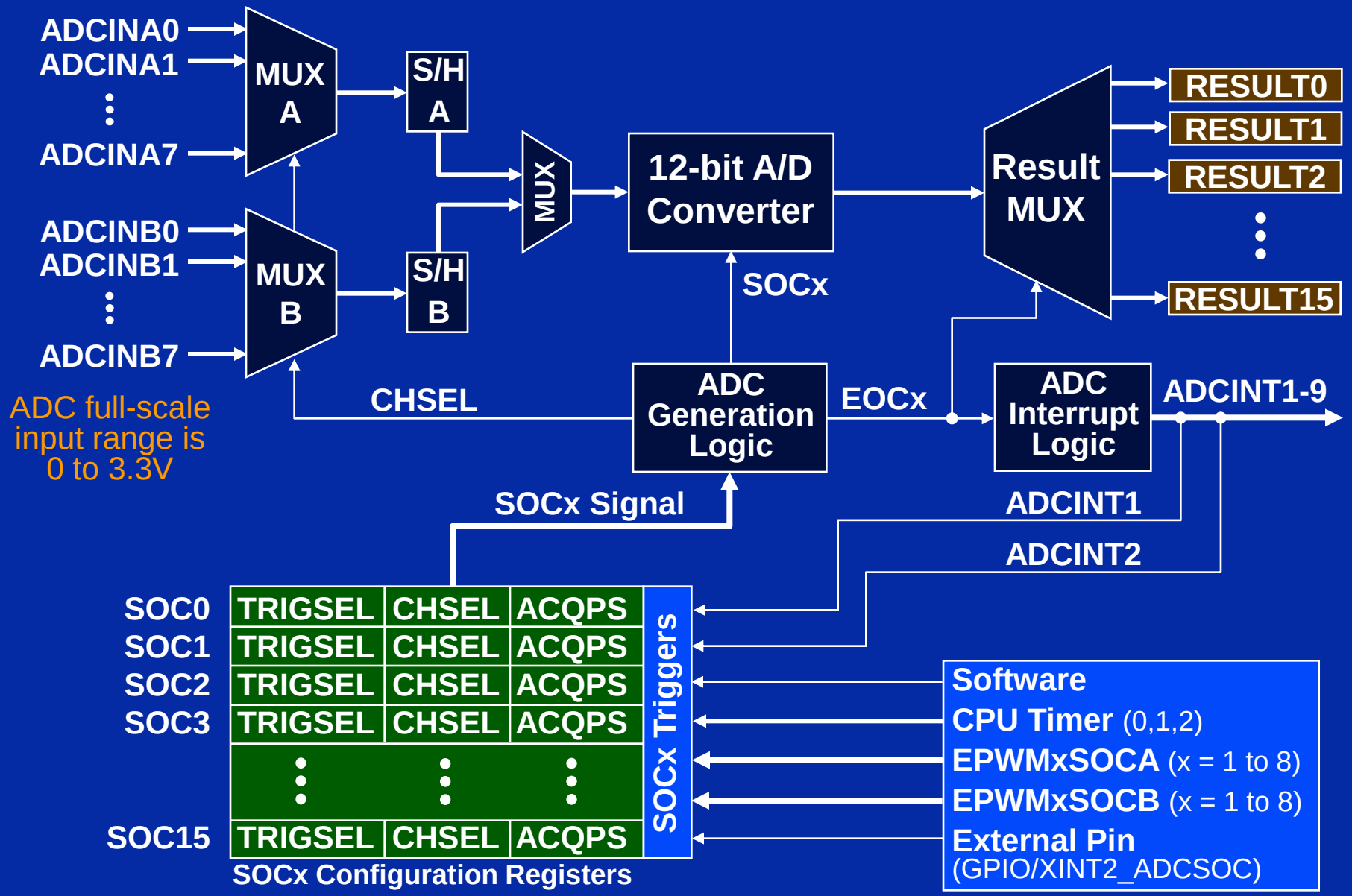
  Part 2

  - Initialize peripheral interrupt expansion (PIE) vectors and use watchdog to generate an interrupt
- Modify, build, and test code using Code Composer Studio

C2000 MCU 1-Day Workshop Outline

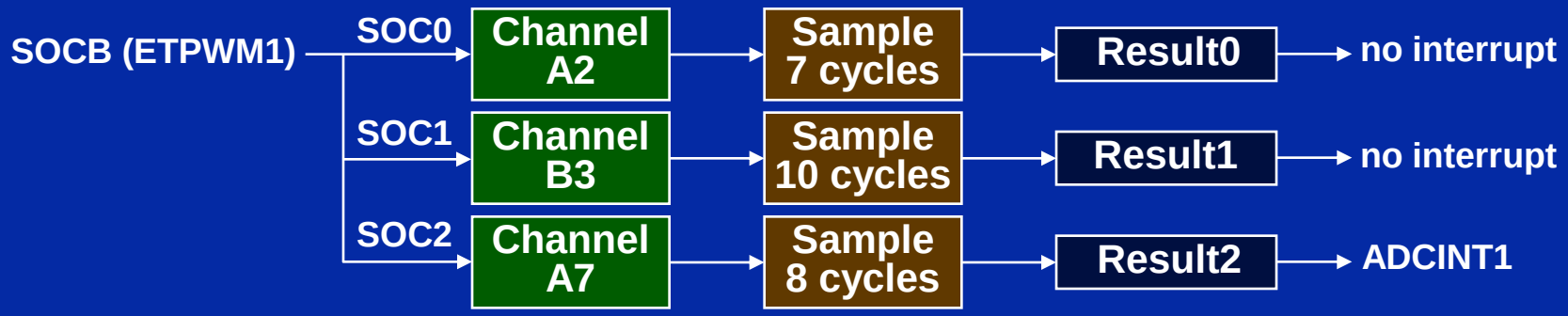
- **Workshop Introduction**
- **Architecture Overview**
- **Programming Development Environment**
  - *Lab: Linker command file*
- **Peripheral Register Header Files**
- **Reset, Interrupts and System Initialization**
  - *Lab: Watchdog and interrupts*
- **Control Peripherals**
  - *Lab: Generate and graph a PWM waveform*
- **Flash Programming**
  - *Lab: Run the code from flash memory*
- **The Next Step…**

ADC Module Block Diagram

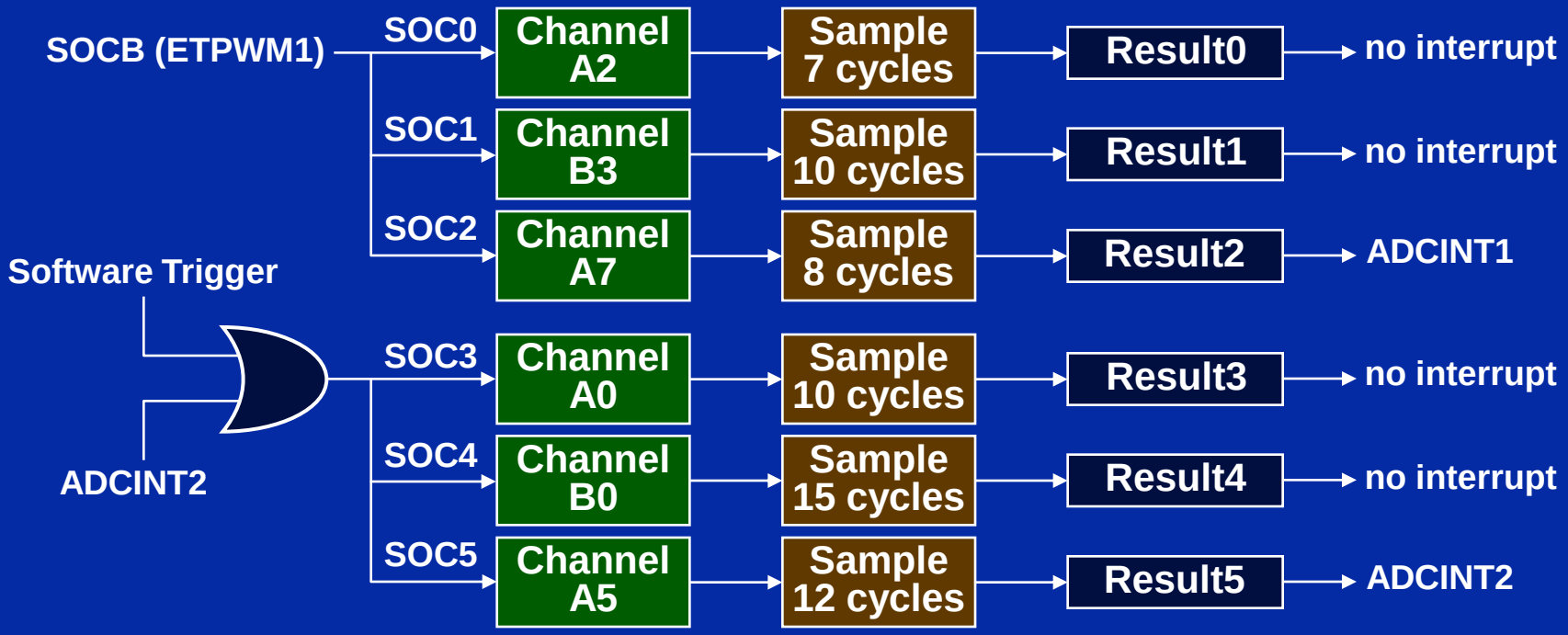


Example – ADC Triggering (1 of 2)

Sample A2 → B3 → A7 when ePWM1 SOCB is generated and then generate ADCINT1:

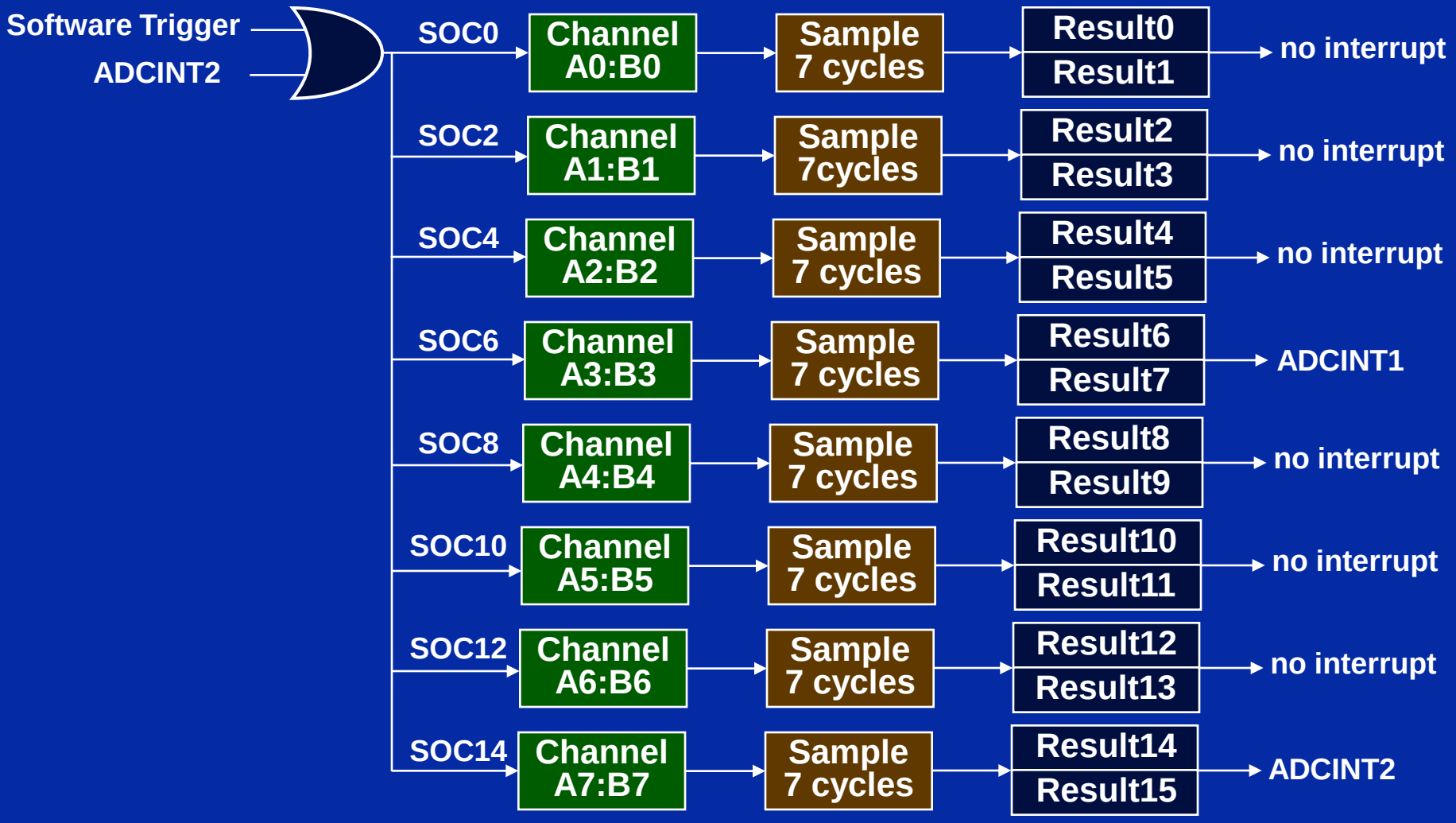


As above, but also sample A0 → B0 → A5 continuously and generate ADCINT2:

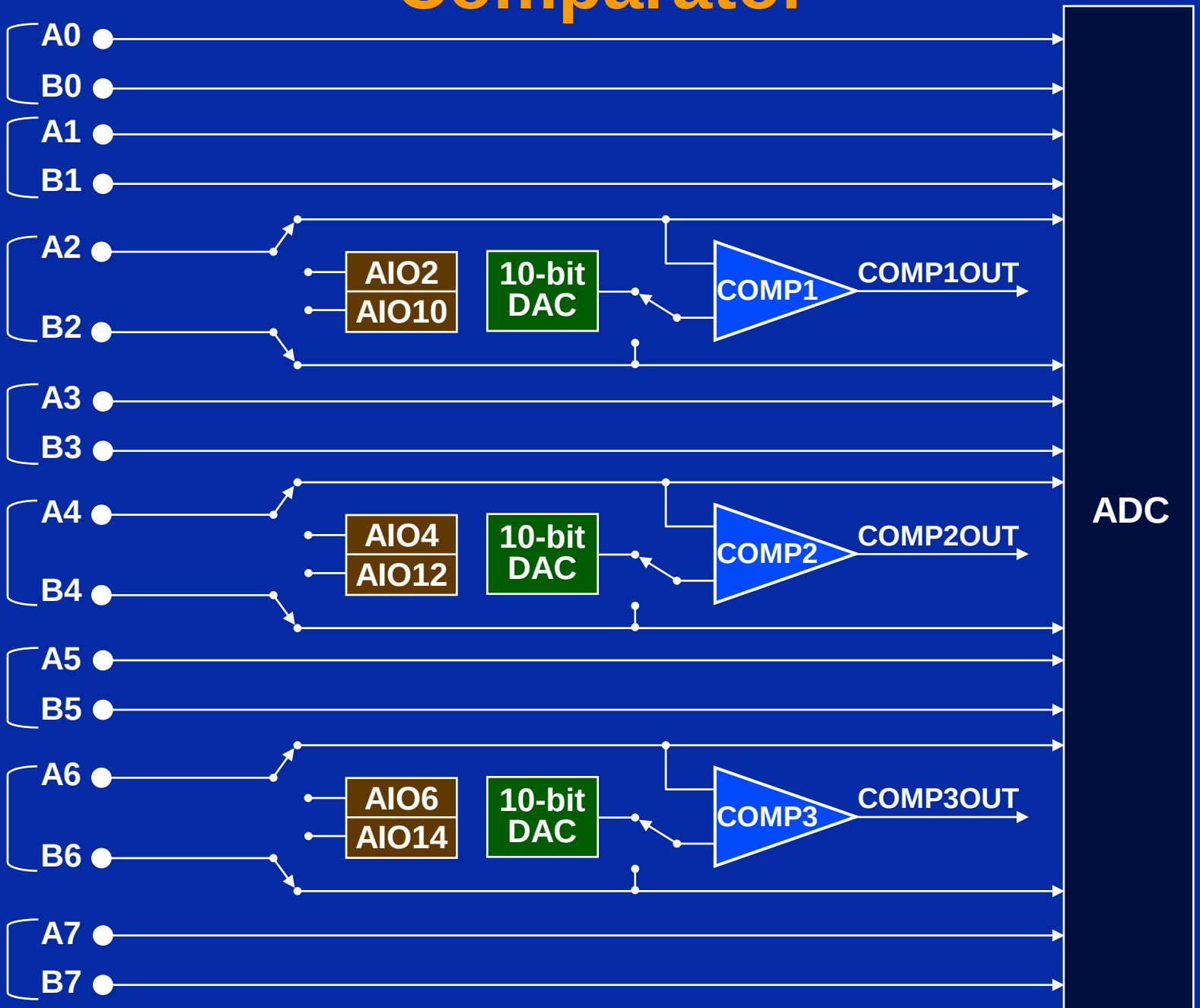


Example – ADC Triggering (2 of 2)

Sample all channels continuously and provide Ping-Pong interrupts to CPU/system:



Comparator


A0
B0
A1
B1
A2
B2
AIO2
AIO10
10-bit DAC
COMP1
COMP1OUT
A3
B3
A4
B4
AIO4
AIO12
10-bit DAC
COMP2
COMP2OUT
A5
B5
A6
B6
AIO6
AIO14
10-bit DAC
COMP3
COMP3OUT
A7
B7
ADC

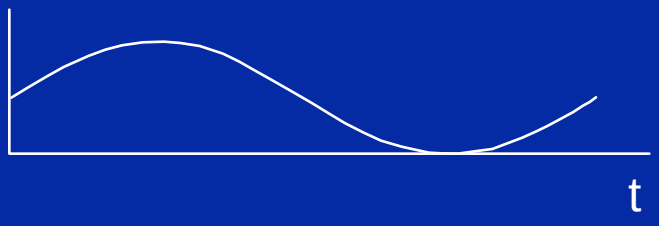
ADC Control Registers (file: Adc.c)

- ADCTRL1 (ADC Control Register 1)
  - module reset, ADC enable
  - busy/busy channel
  - reference select
  - Interrupt generation control
- ADCSOCxCTL (SOC0 to SOC15 Control Registers)
  - trigger source
  - channel
  - acquisition sampling window
- ADCINTSOCSELx (Interrupt SOC Selection 1 and 2 Registers)
  - selects ADCINT1 / ADCINT2 trigger for SOCx
- ADCSAMPLEMODE (Sampling Mode Register)
  - sequential sampling / simultaneous sampling
- INTSELxNy (Interrupt x and y Selection Registers)
  - EOC0 – EOC15 source select for ADCINT1-9
- ADCRESULTx (ADC Result 0 to 15 Registers)

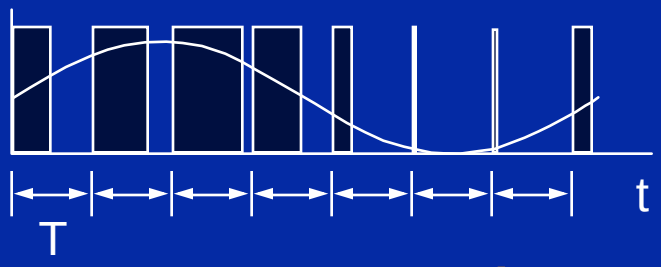
Note: refer to the reference guide for a complete listing of registers

What is Pulse Width Modulation?

- PWM is a scheme to represent a signal as a sequence of pulses
  - fixed carrier frequency
  - fixed pulse amplitude
  - pulse width proportional to instantaneous signal amplitude
  - PWM energy ≈ original signal energy



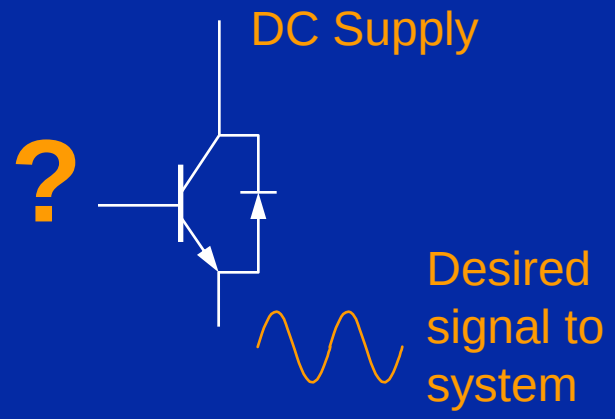
Original Signal



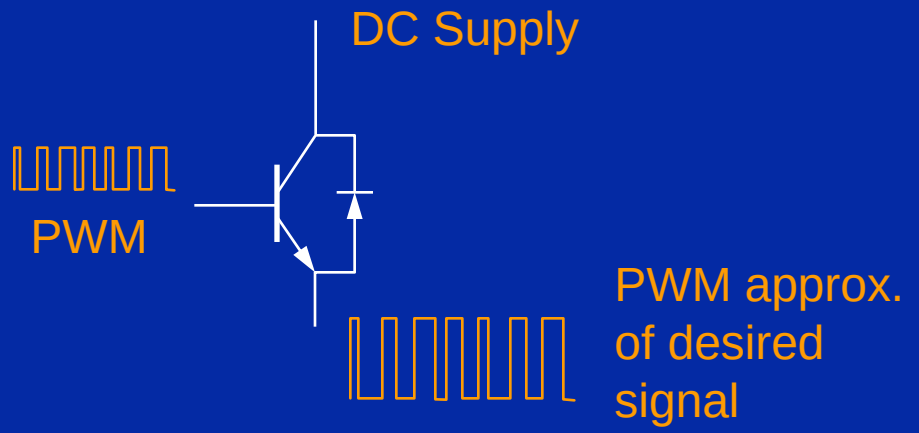
PWM representation

Why use PWM with Power Switching Devices?

- Desired output currents or voltages are known
- Power switching devices are transistors
  - Difficult to control in proportional region
  - Easy to control in saturated region
- PWM is a digital signal ⇒ easy for MCU to output

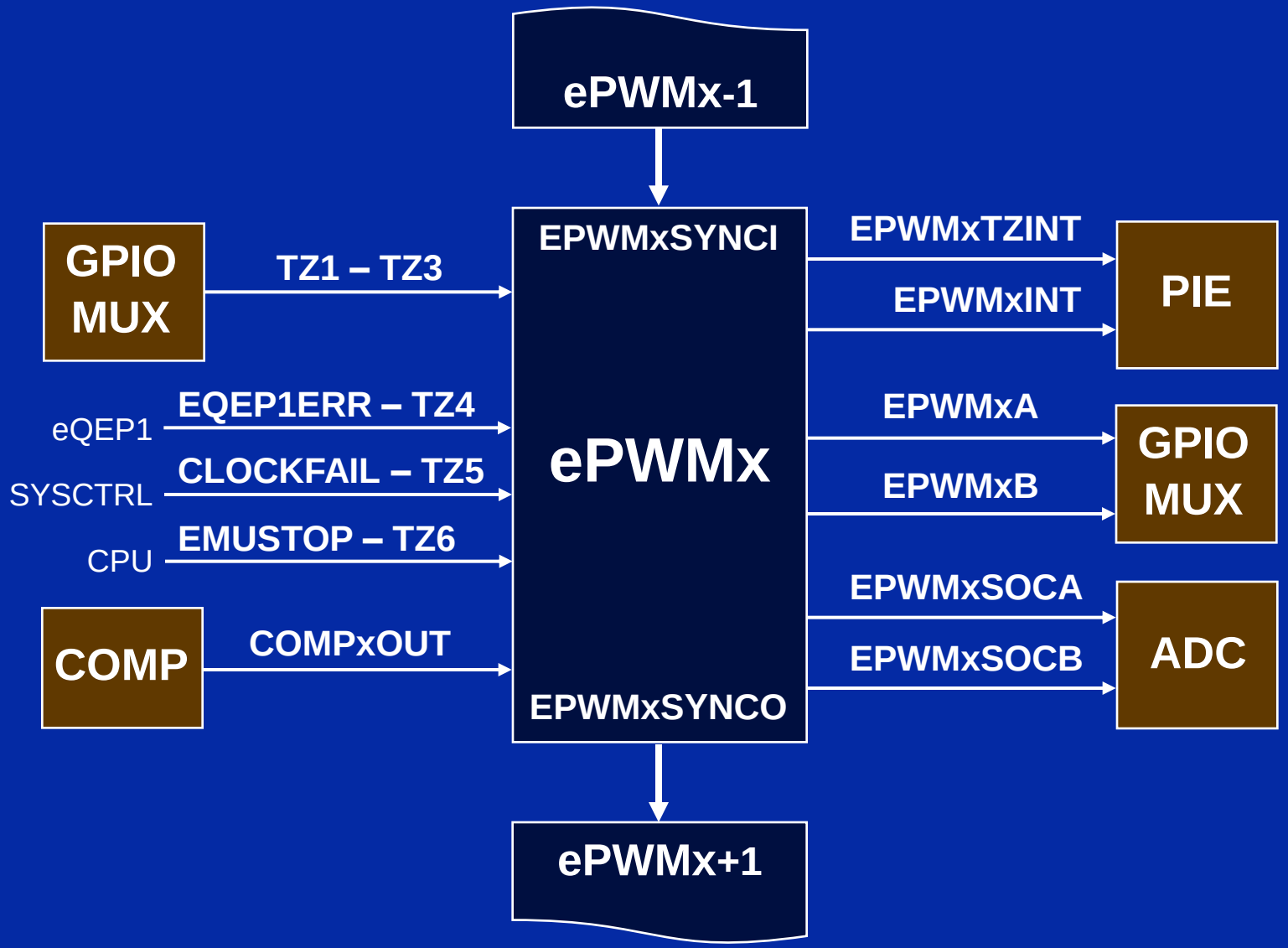


Unknown Gate Signal

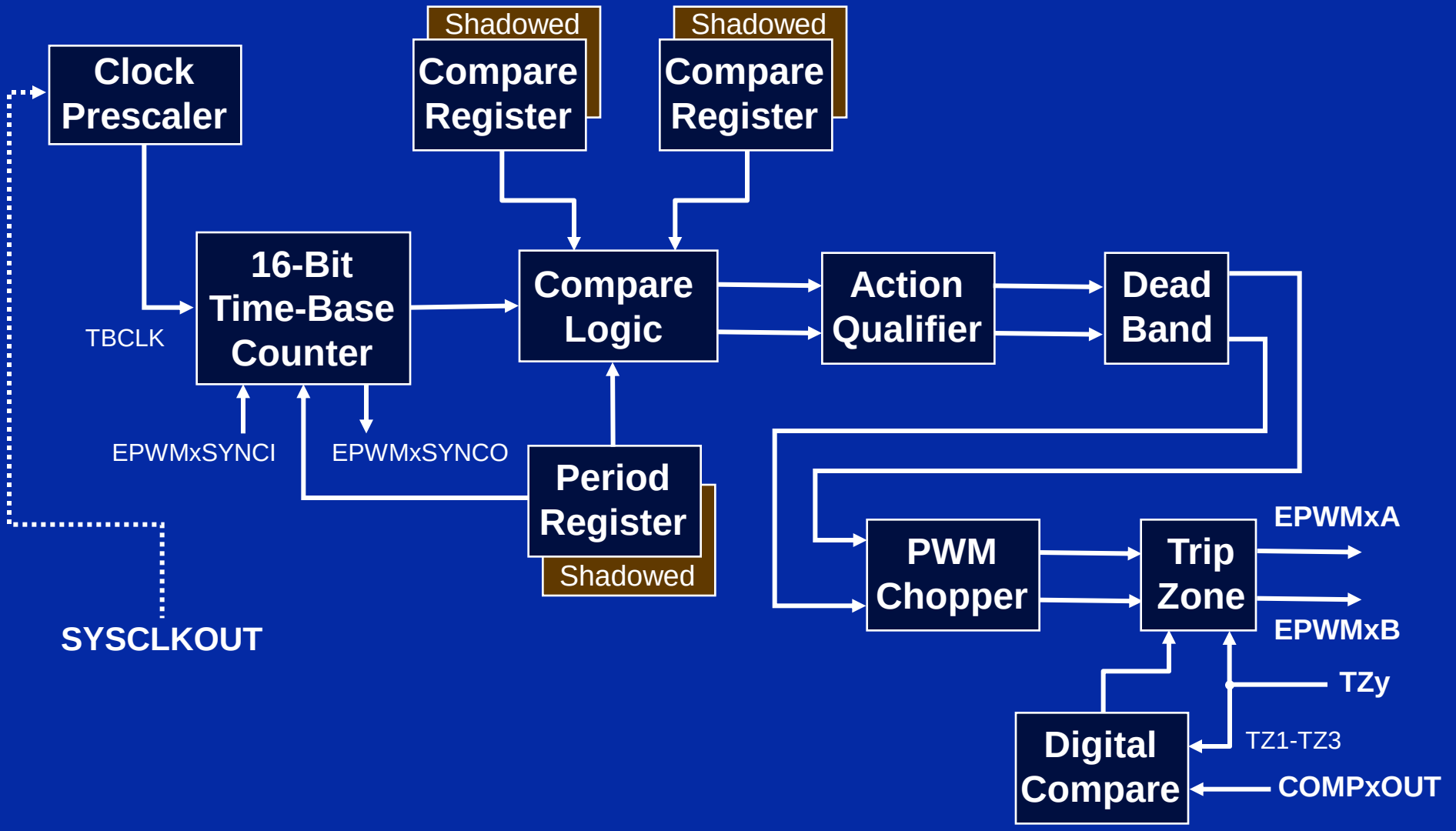


Gate Signal Known with PWM

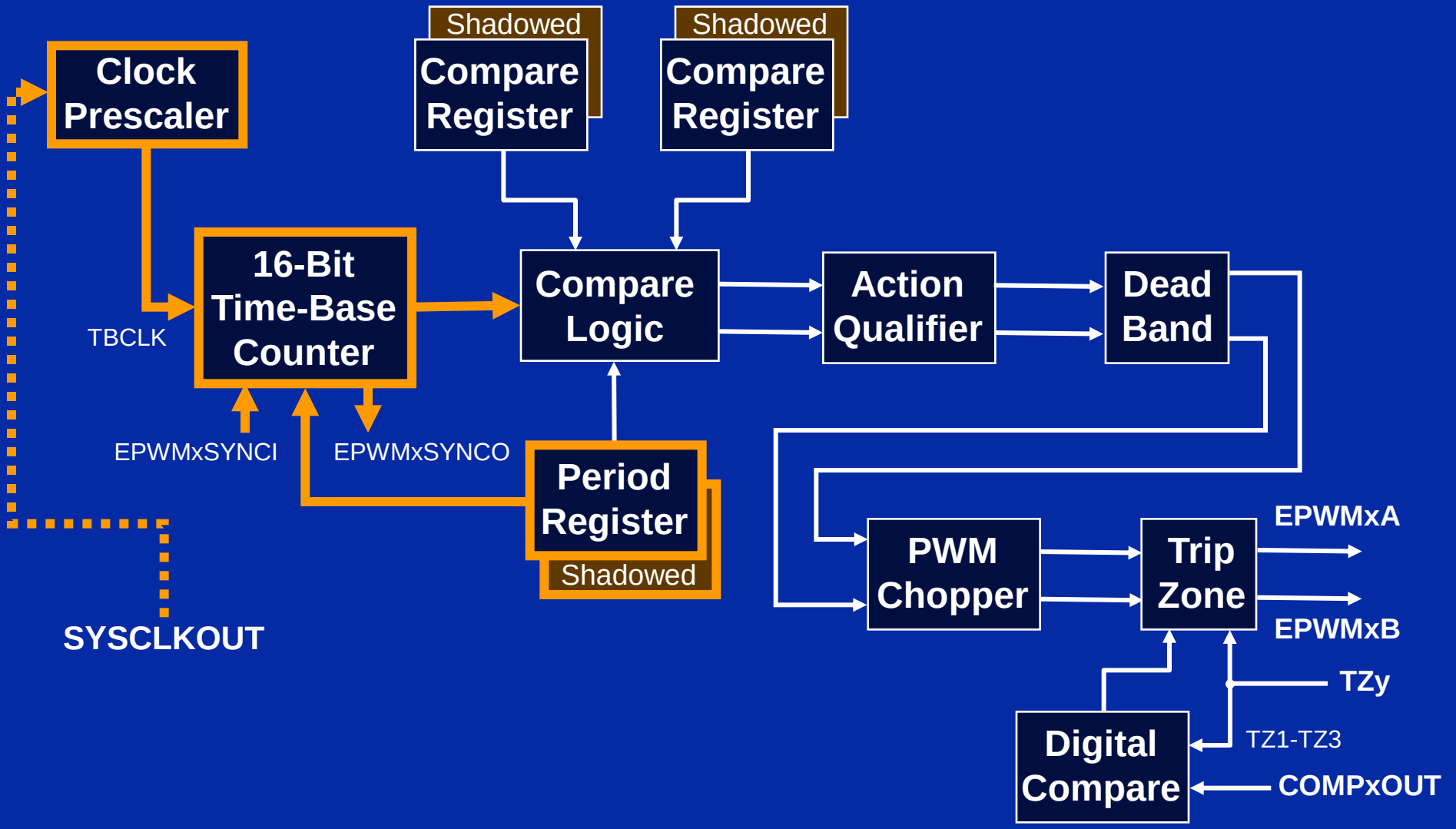
ePWM Module Signals and Connections



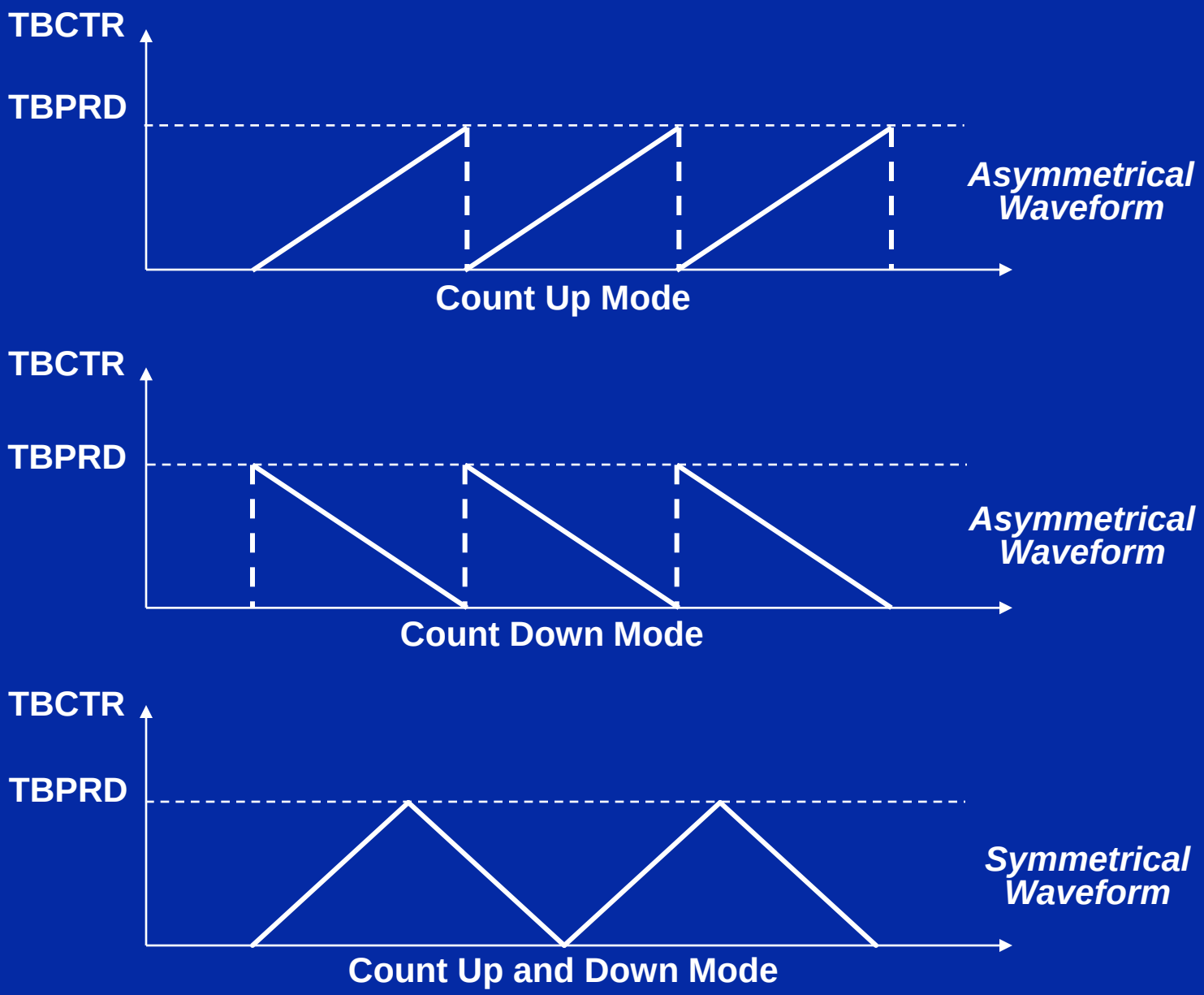
ePWM Block Diagram



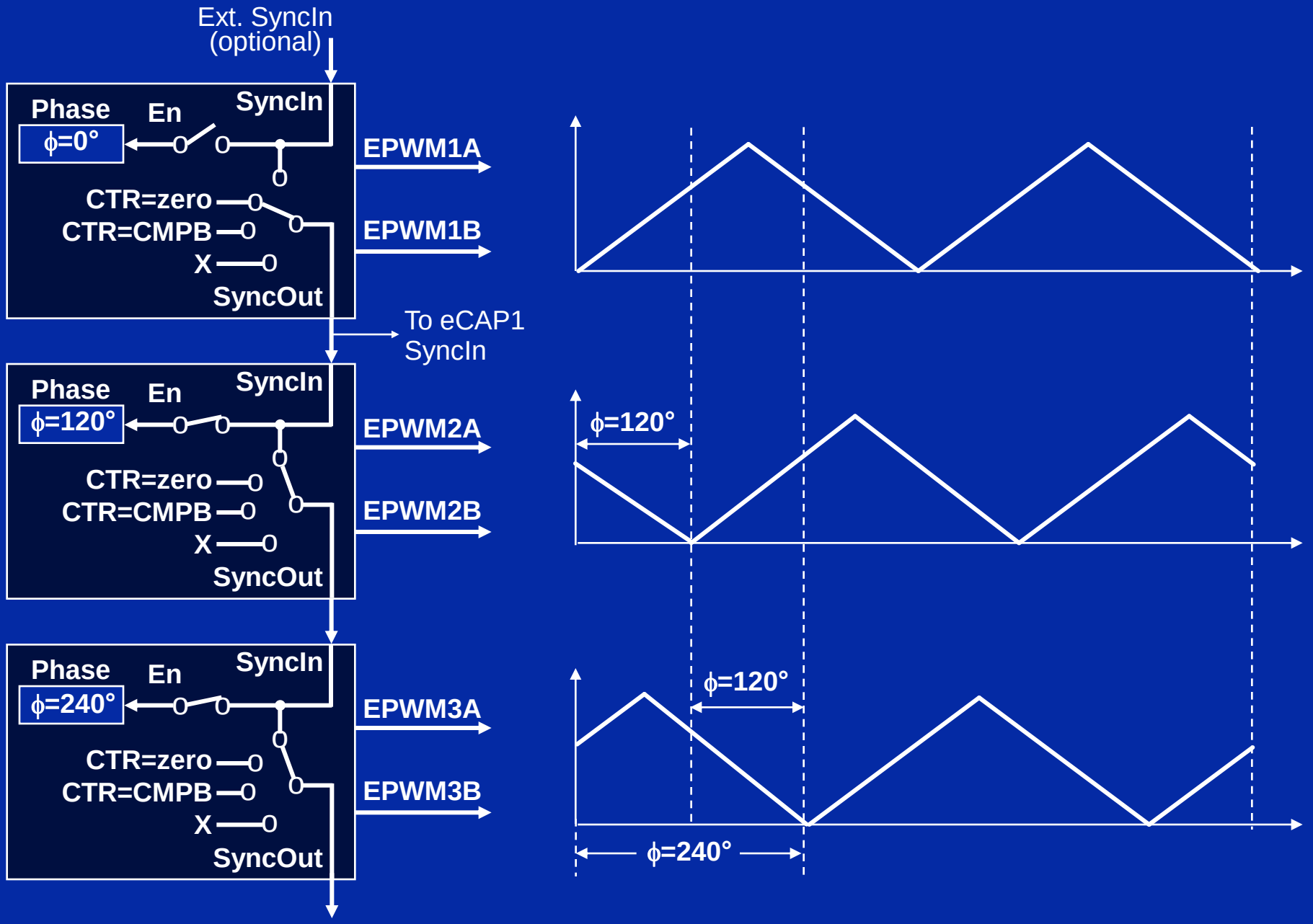
ePWM Time-Base Sub-Module



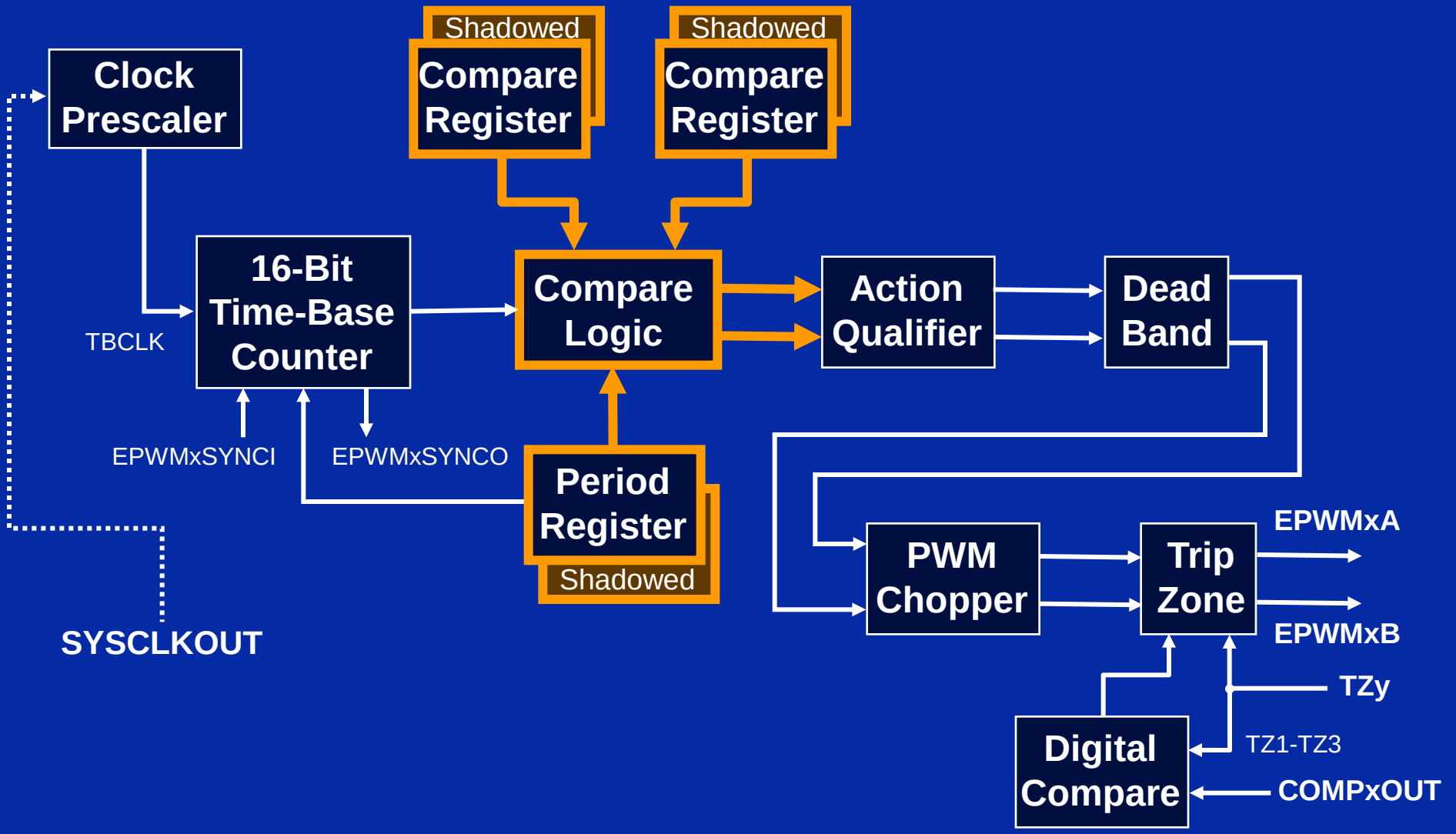
ePWM Time-Base Count Modes



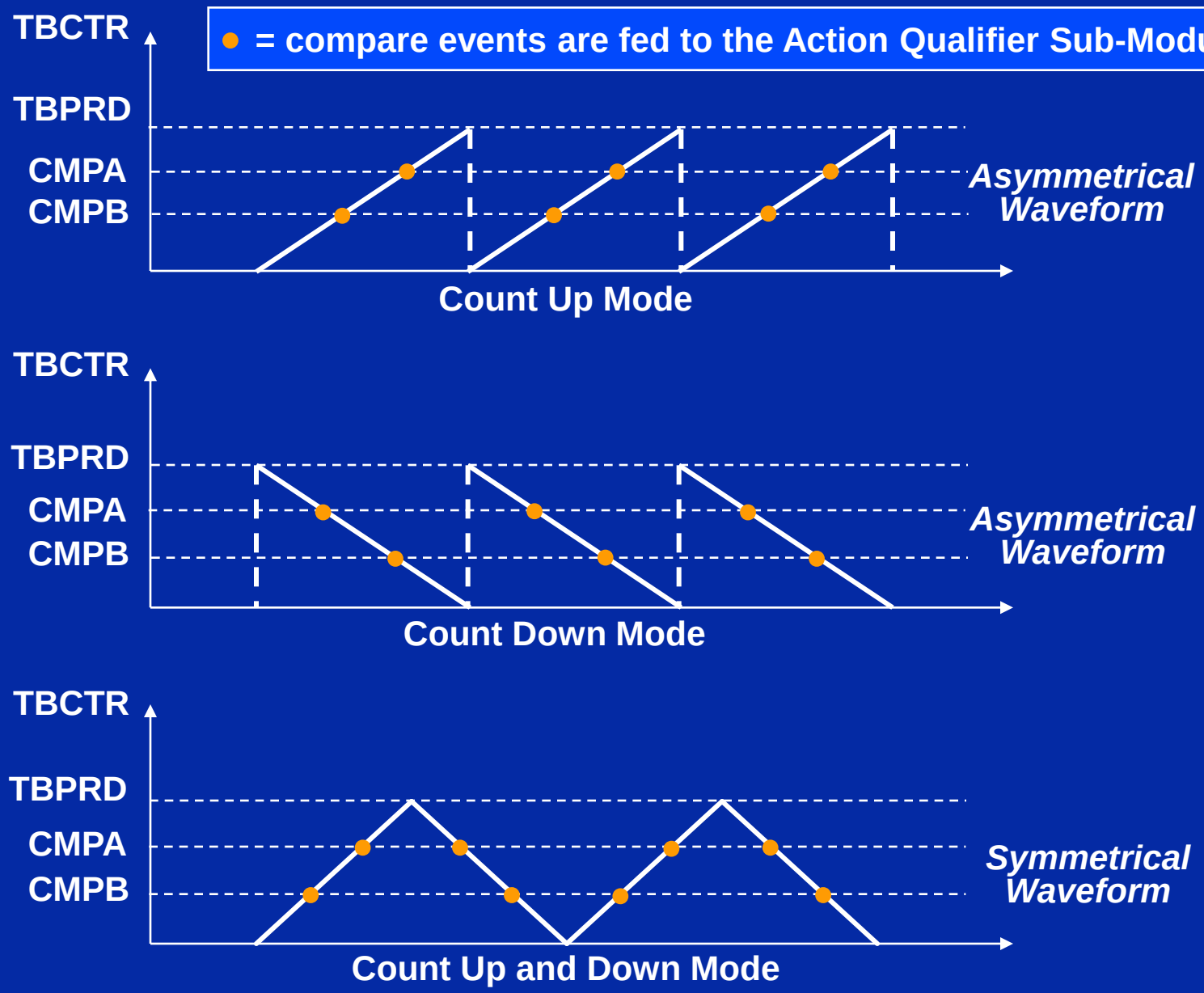
ePWM Phase Synchronization



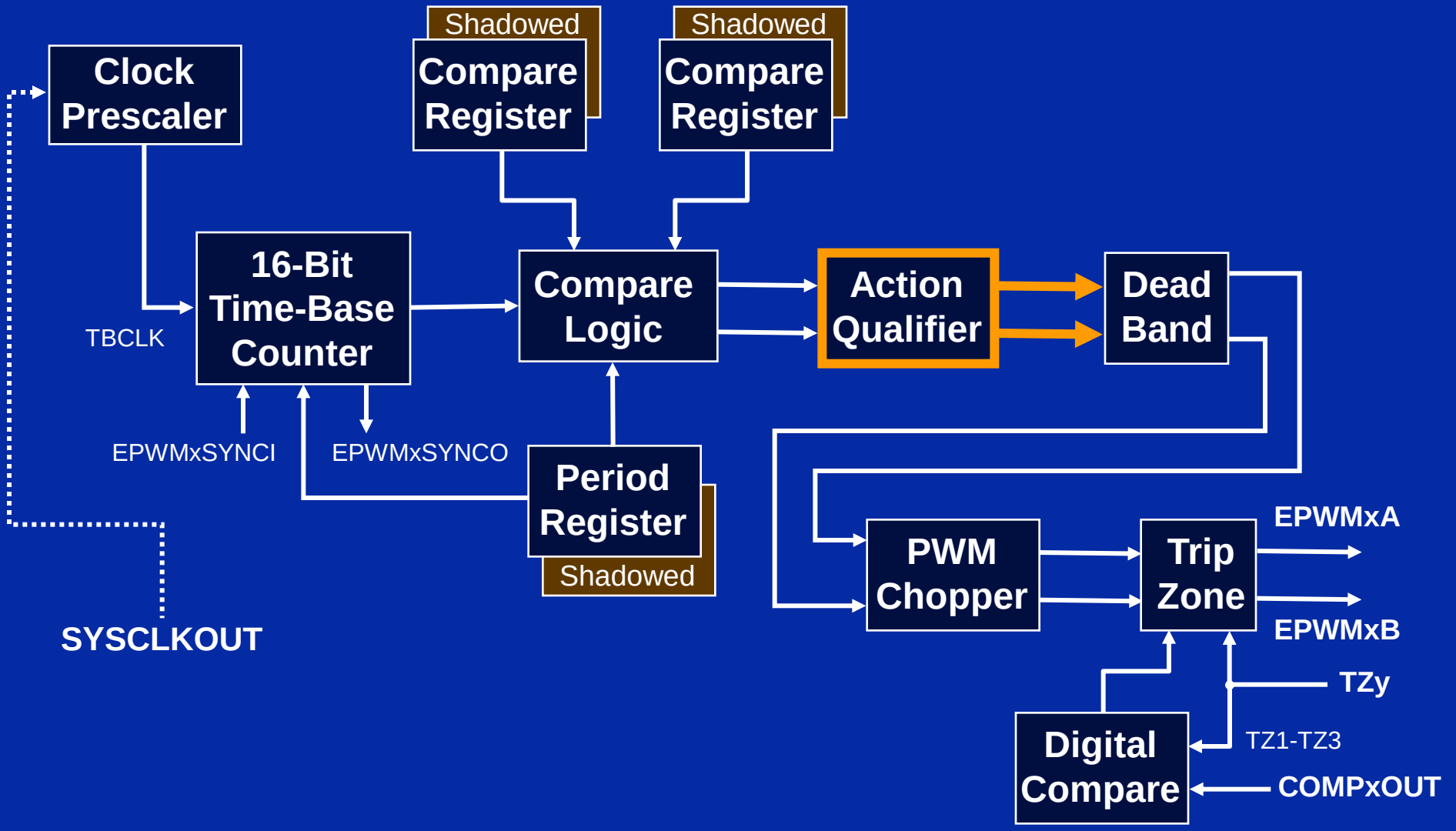
ePWM Compare Sub-Module



ePWM Compare Event Waveforms



ePWM Action Qualifier Sub-Module



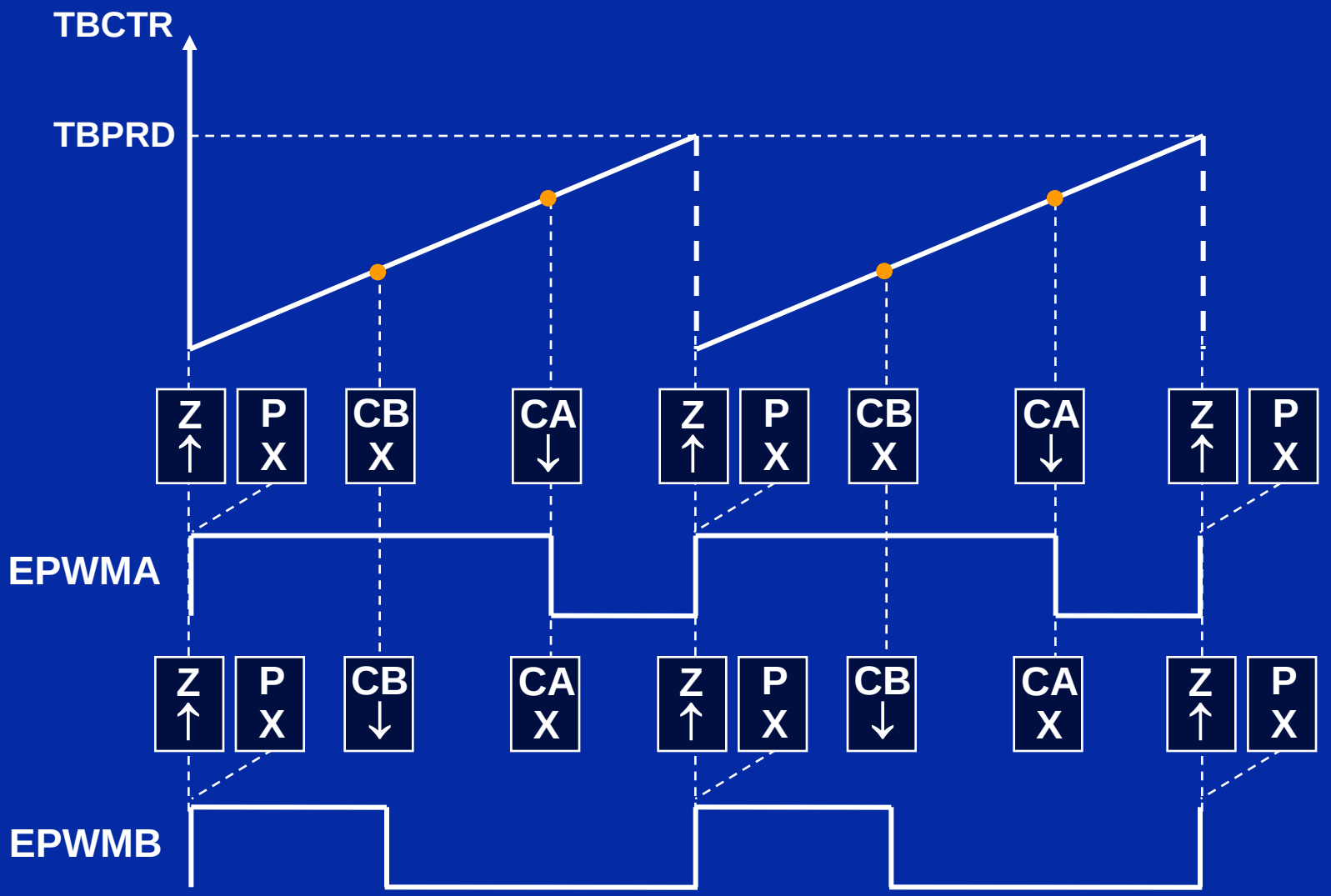
ePWM Action Qualifier Actions

for EPWMA and EPWMB

S/W Force	Time-Base Counter equals:				EPWM Output Actions
	Zero	CMPA	CMPB	TBPRD	
SW X	Z X	CA X	CB X	P X	Do Nothing
SW ↓	Z ↓	CA ↓	CB ↓	P ↓	Clear Low
SW ↑	Z ↑	CA ↑	CB ↑	P ↑	Set High
SW T	Z T	CA T	CB T	P T	Toggle

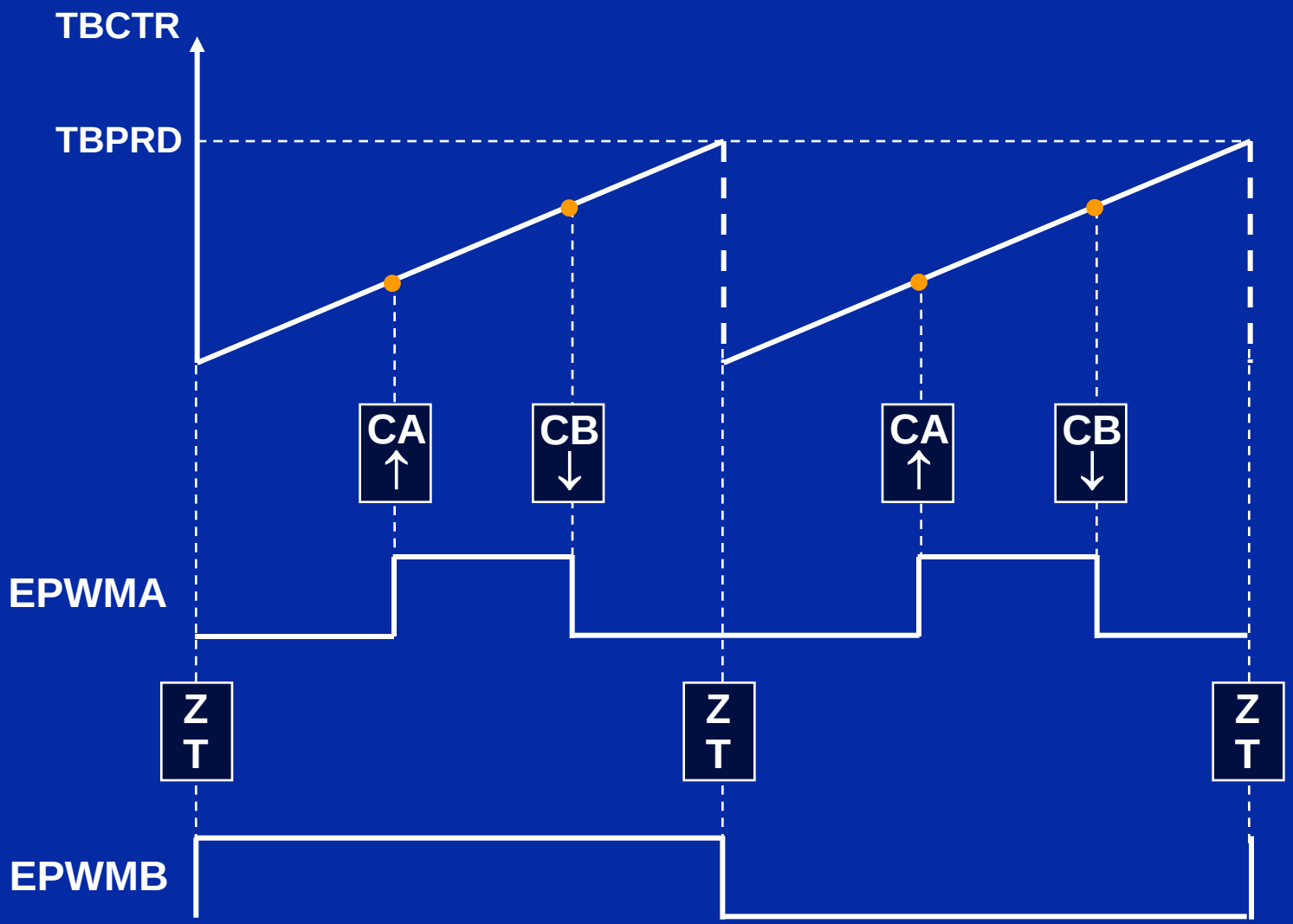
ePWM Count Up Asymmetric Waveform

with Independent Modulation on EPWMA / B



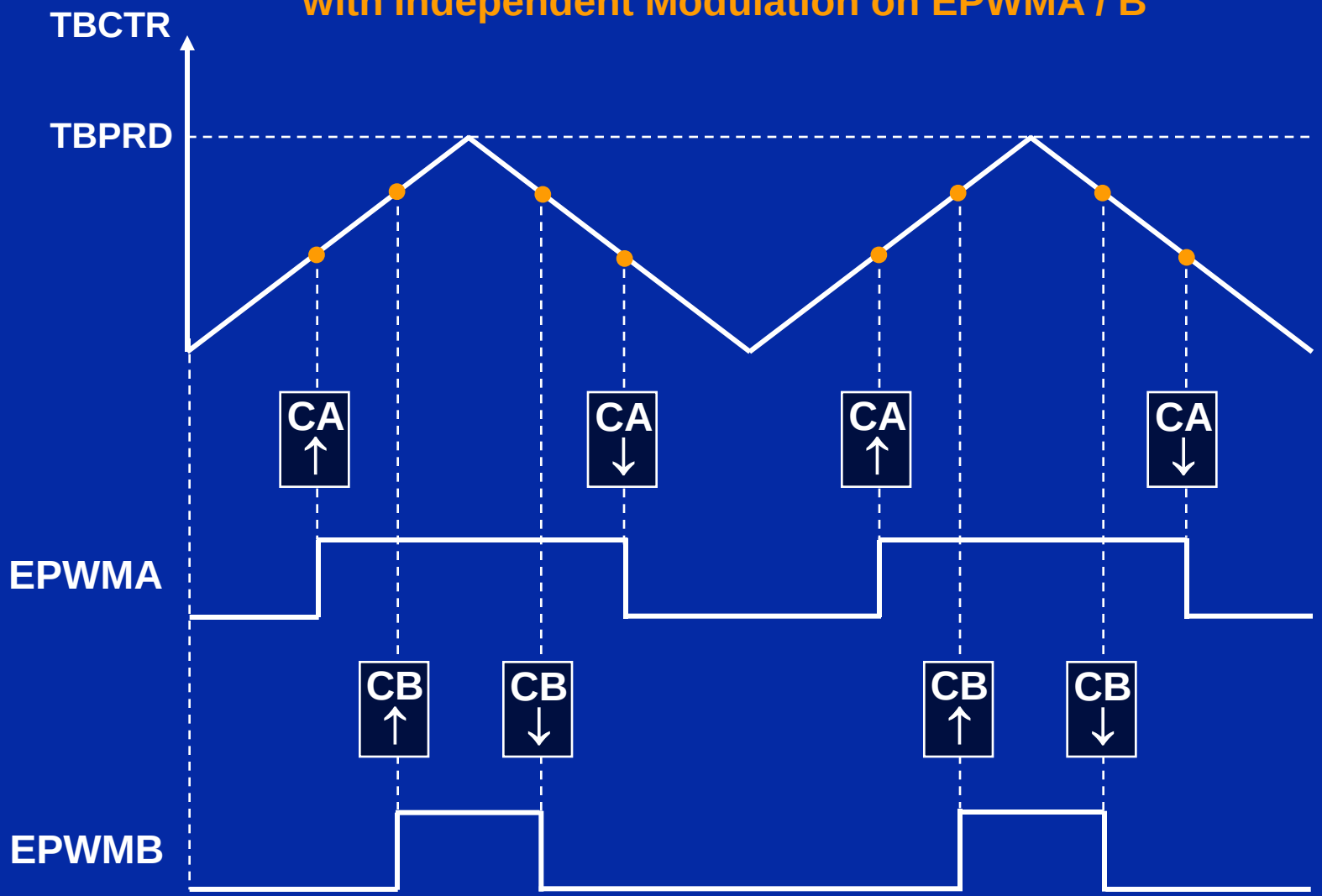
ePWM Count Up Asymmetric Waveform

with Independent Modulation on EPWMA



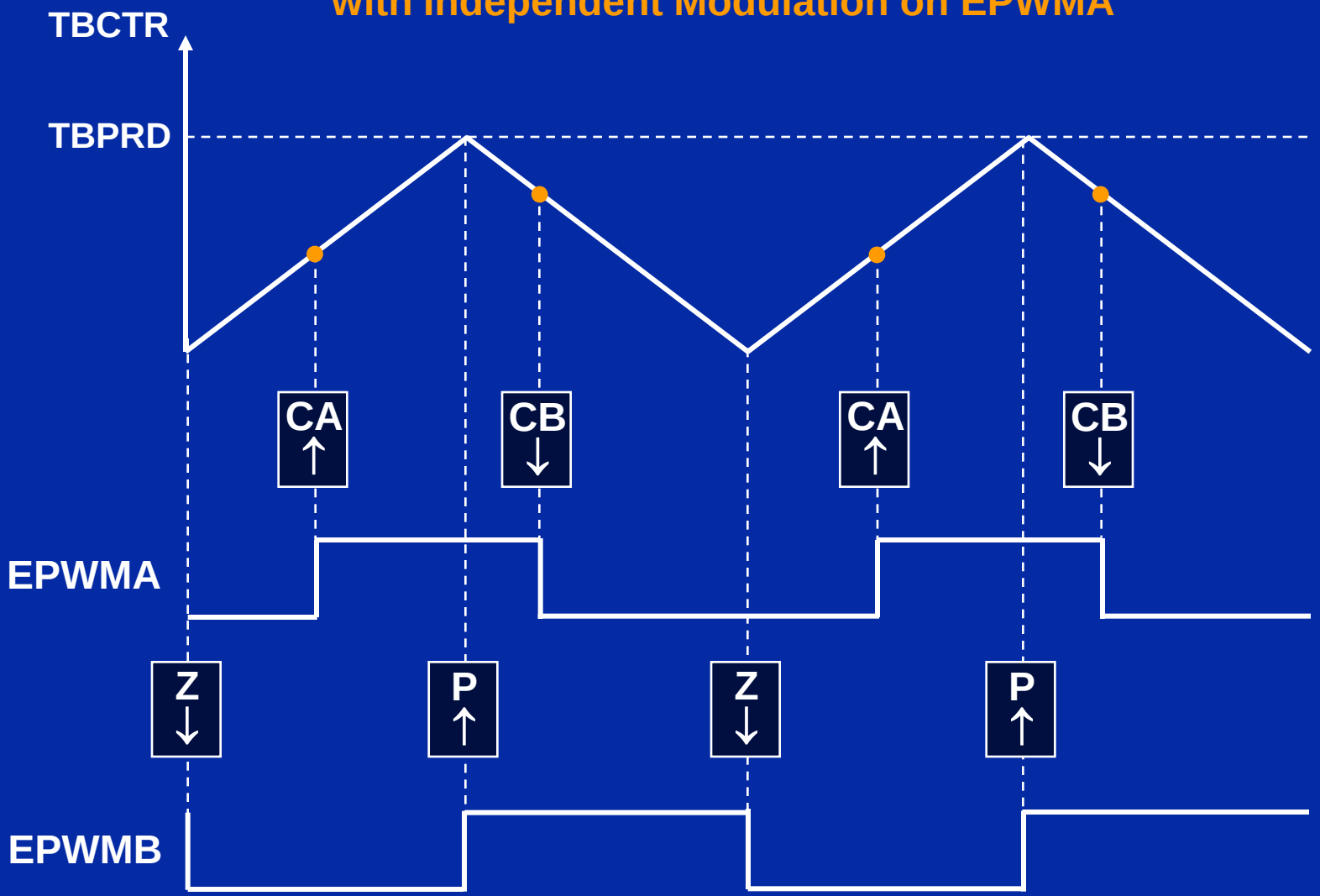
ePWM Count Up-Down Symmetric Waveform

with Independent Modulation on EPWMA / B

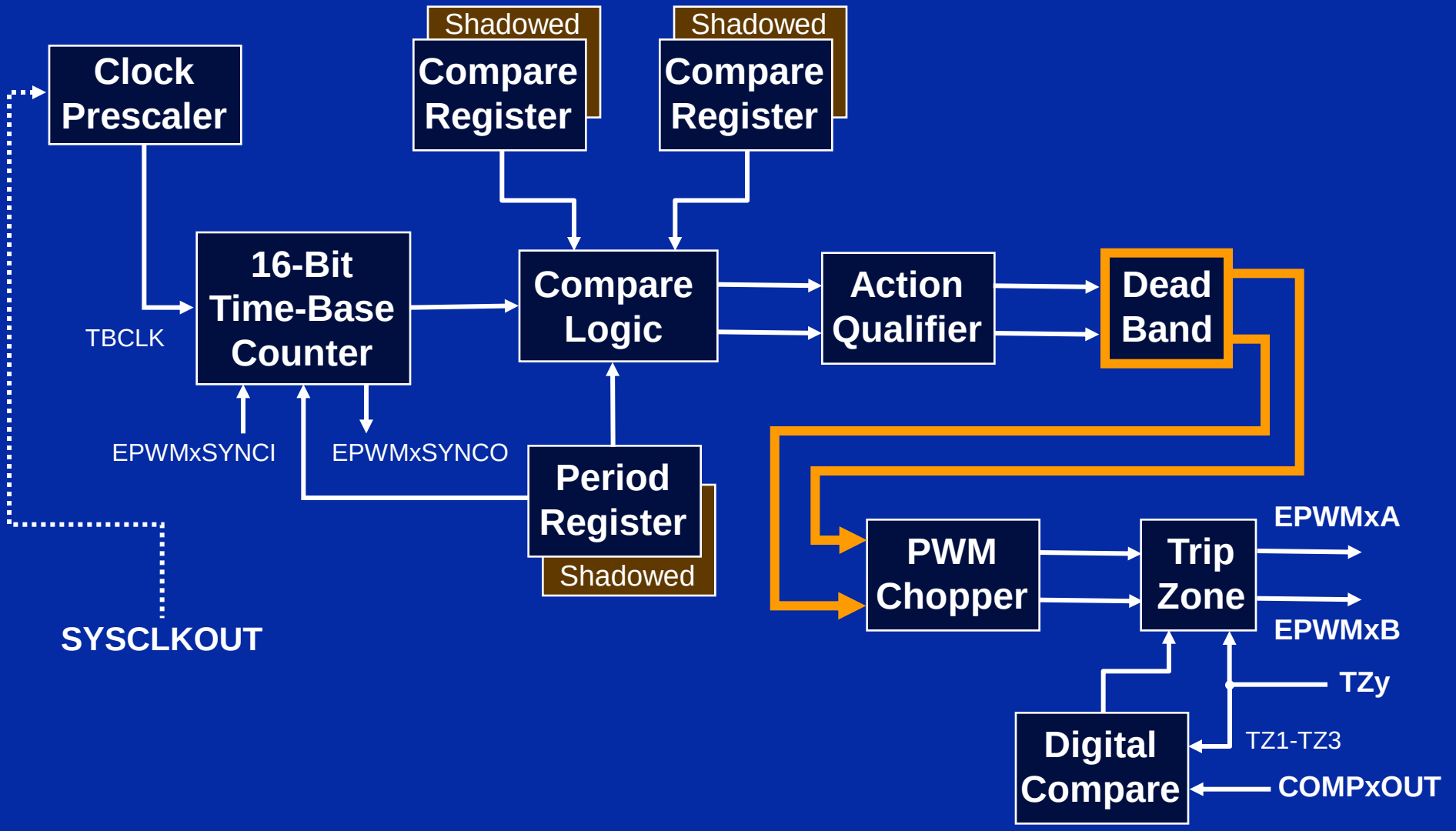


ePWM Count Up-Down Symmetric Waveform

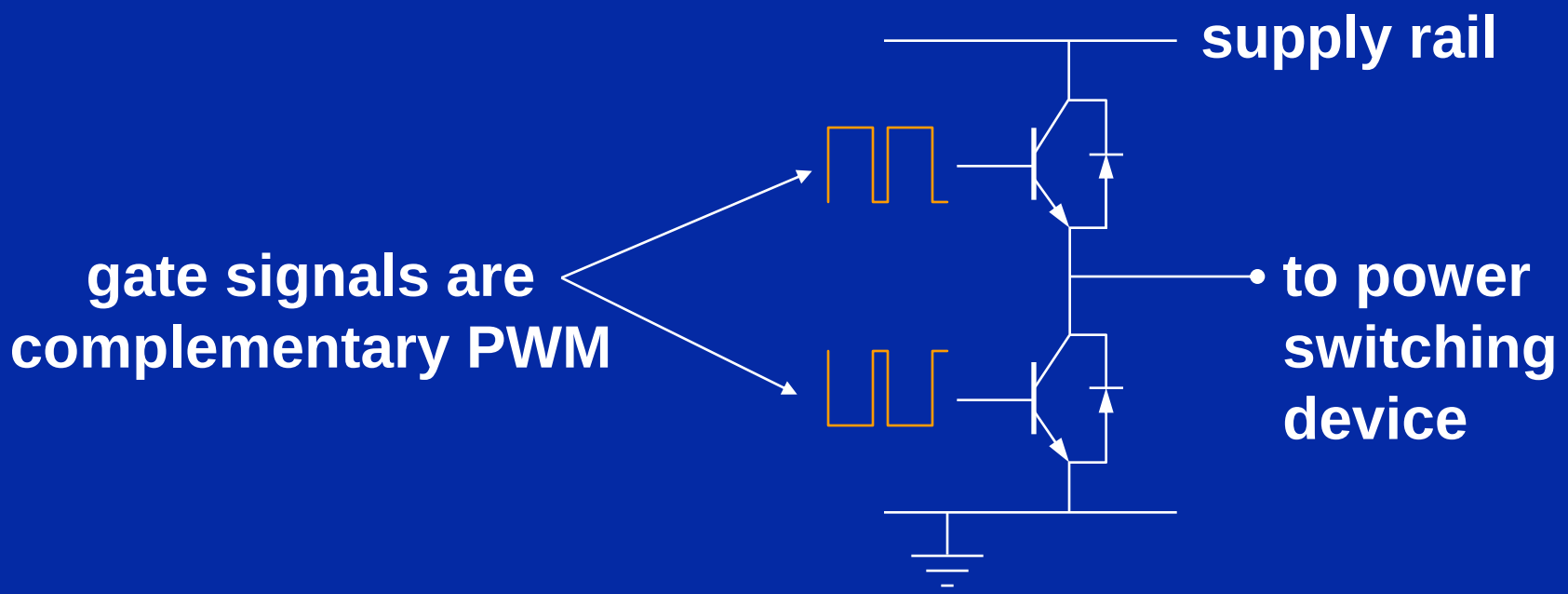
with Independent Modulation on EPWMA



ePWM Dead-Band Sub-Module

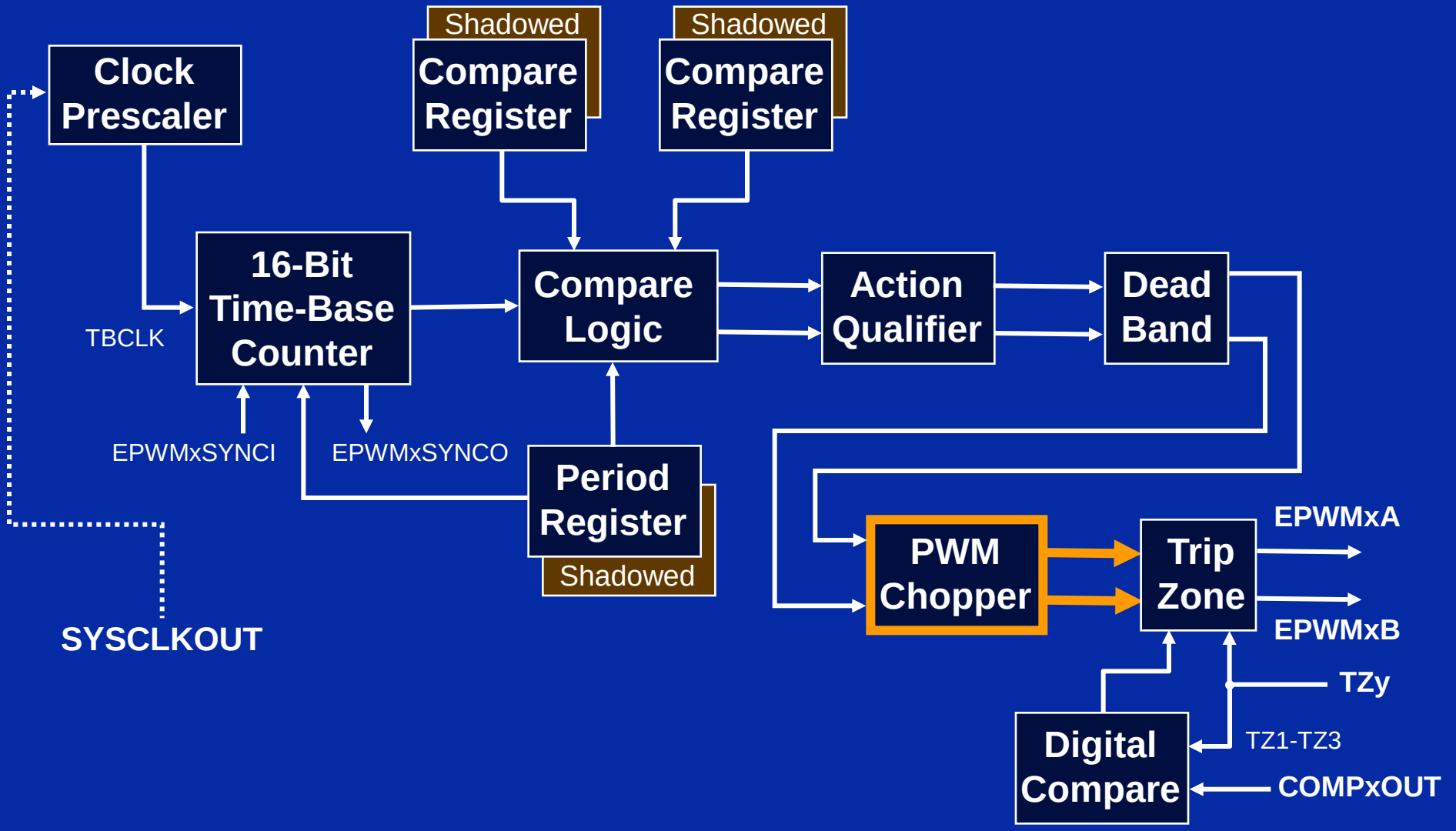


Motivation for Dead-Band

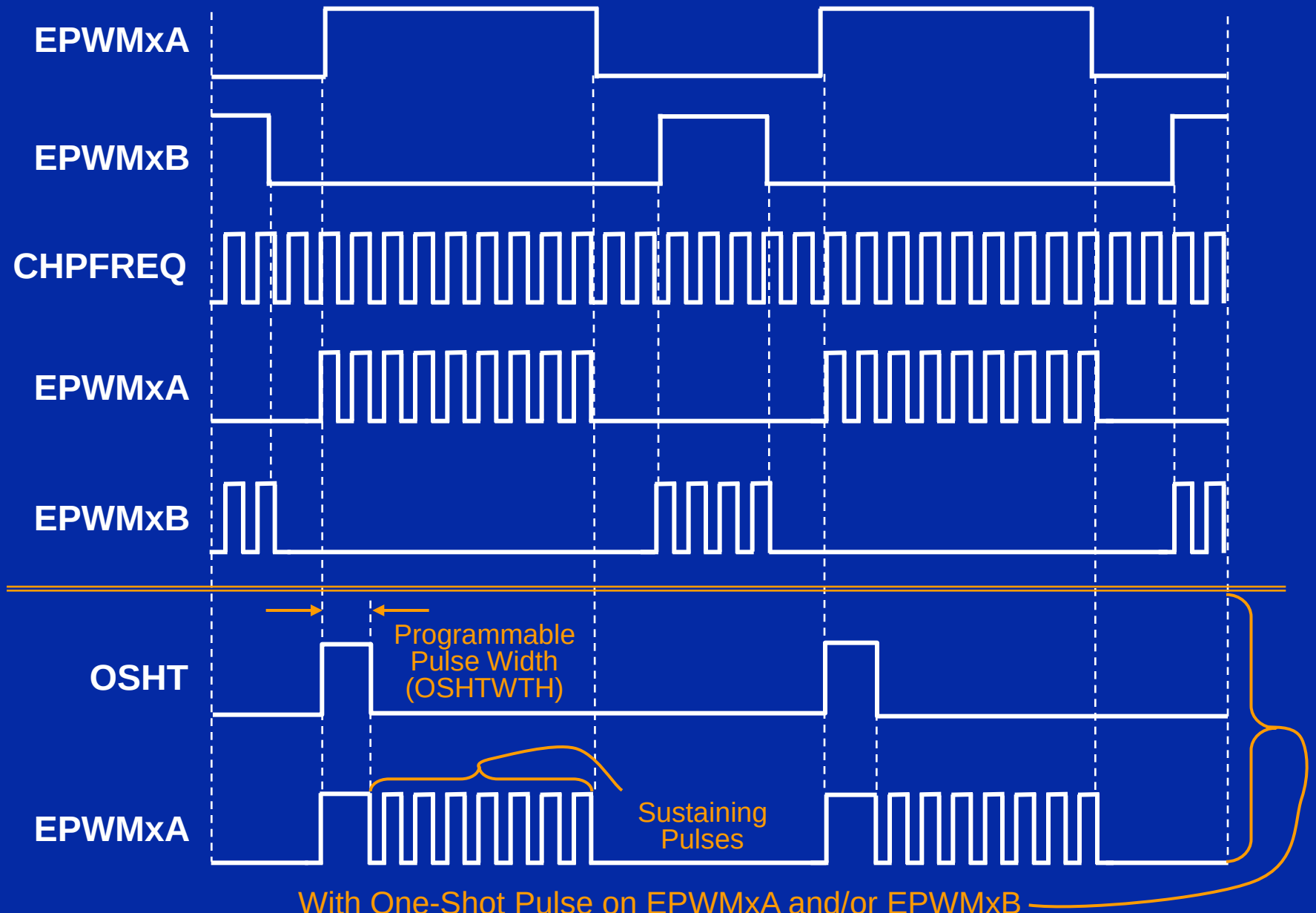


- Transistor gates turn on faster than they shut off
- Short circuit if both gates are on at same time!

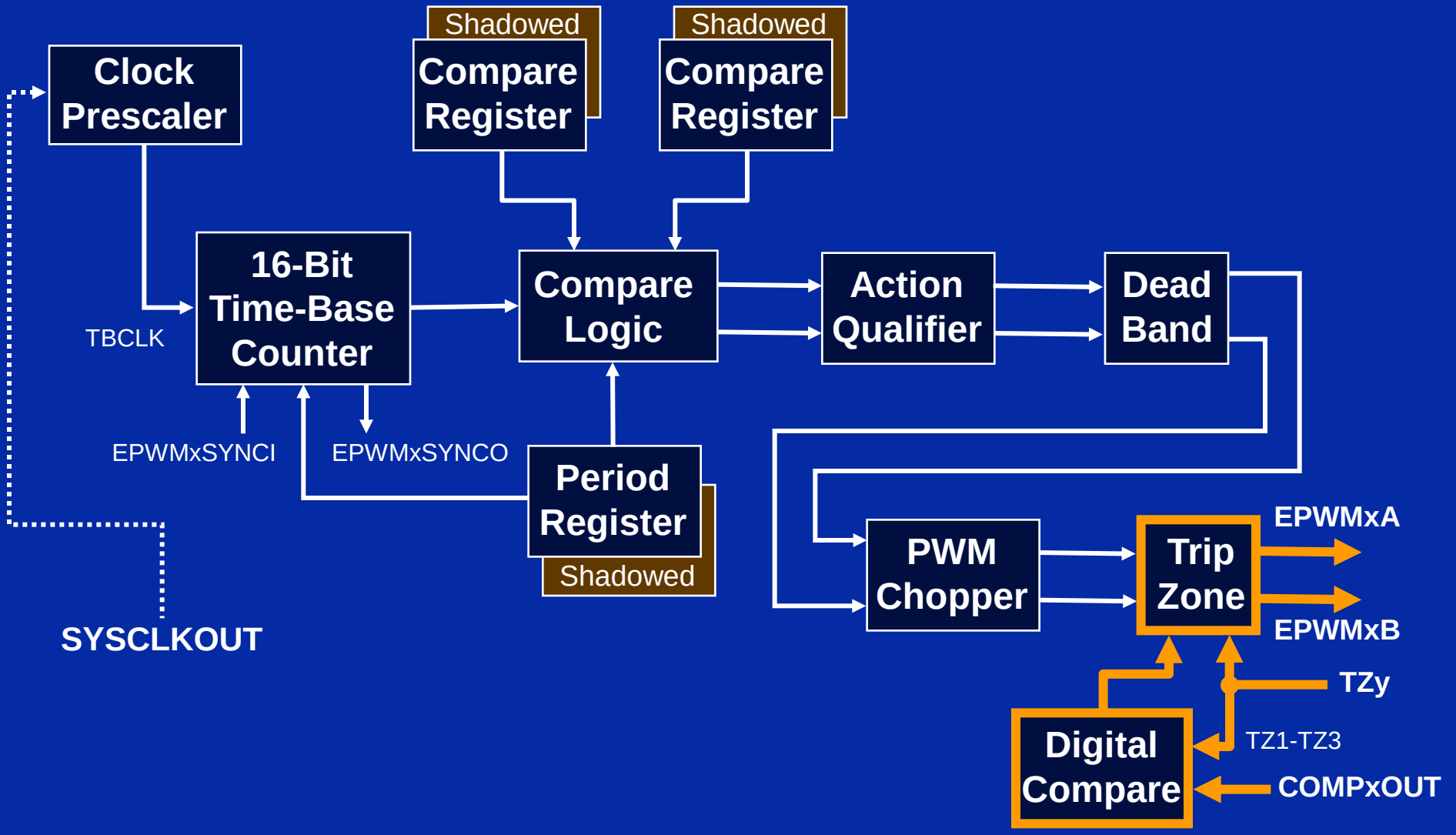
ePWM PWM Chopper Sub-Module



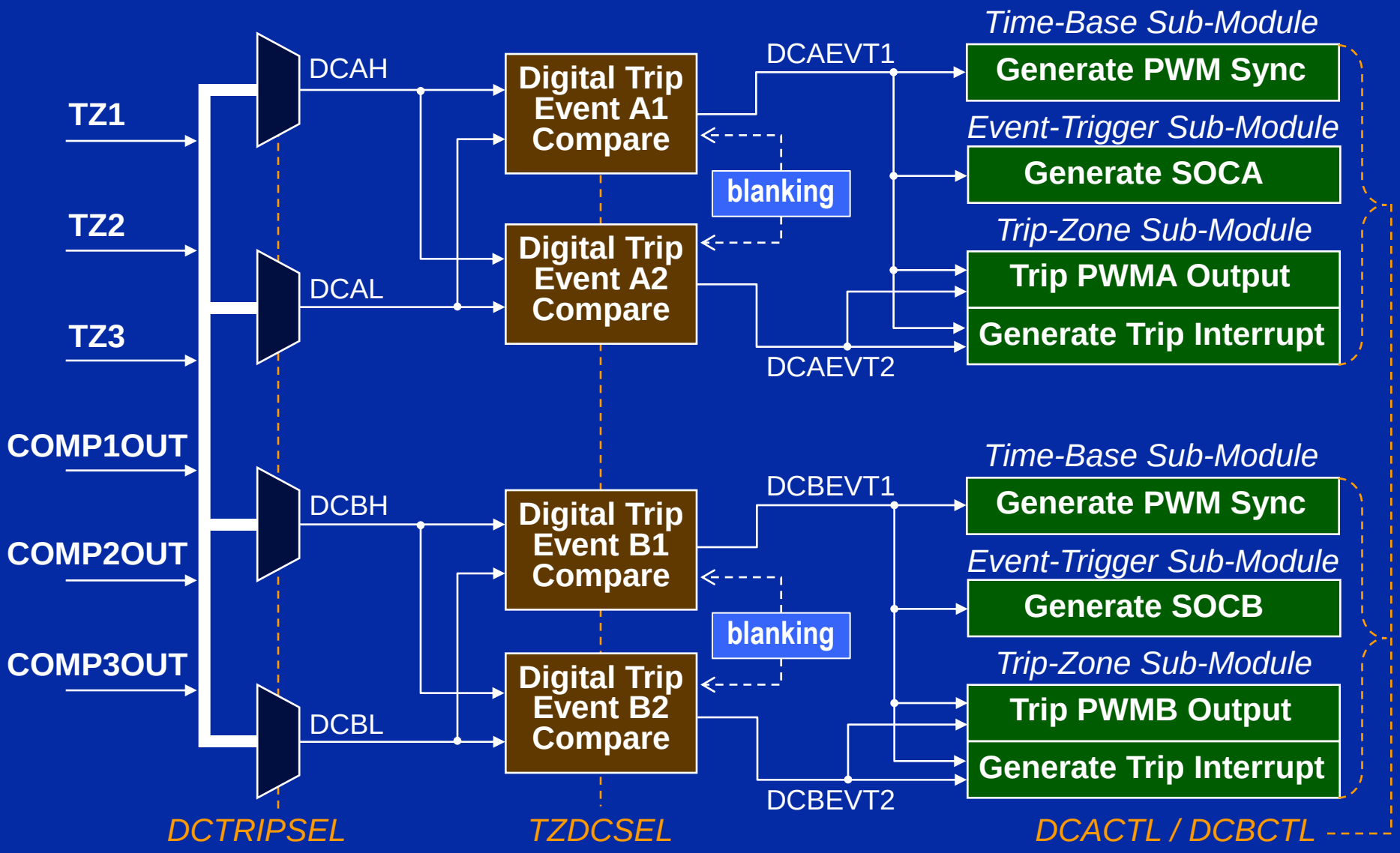
ePWM Chopper Waveform



ePWM Digital Compare and Trip-Zone Sub-Modules



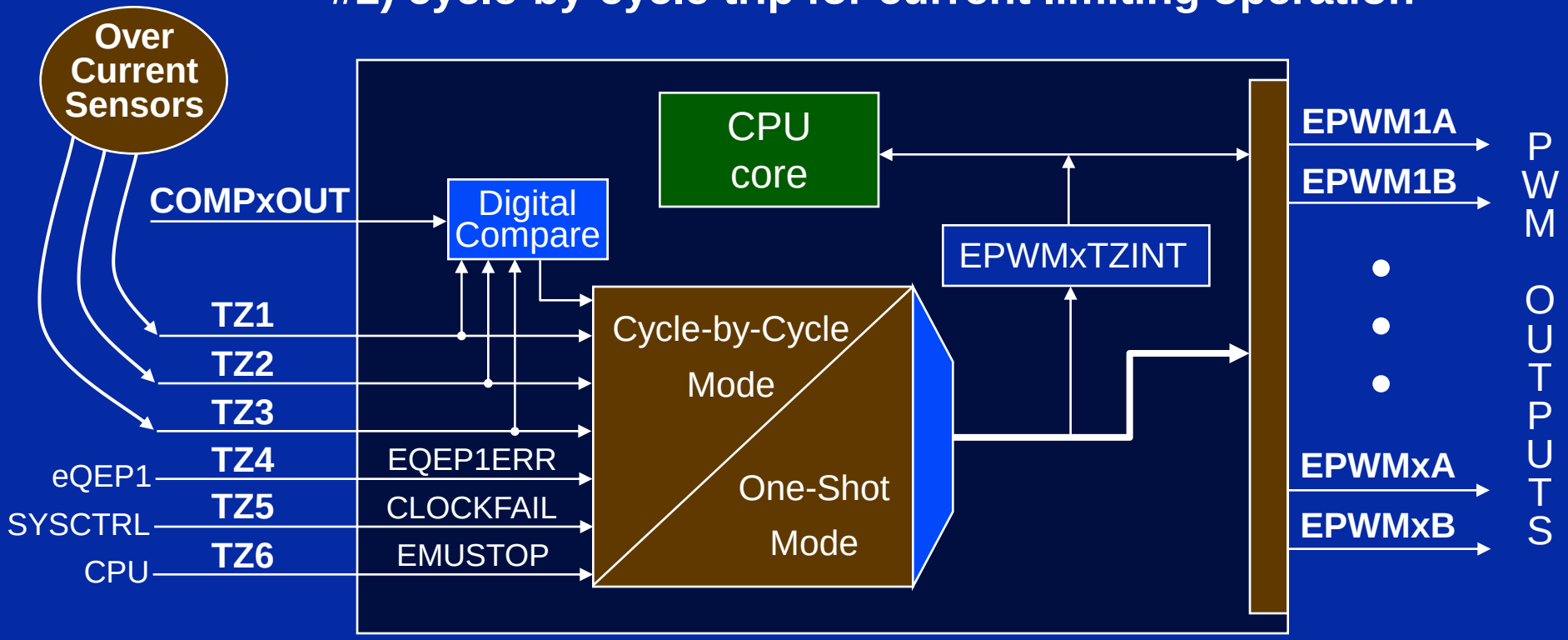
Digital Compare Sub-Module Signals



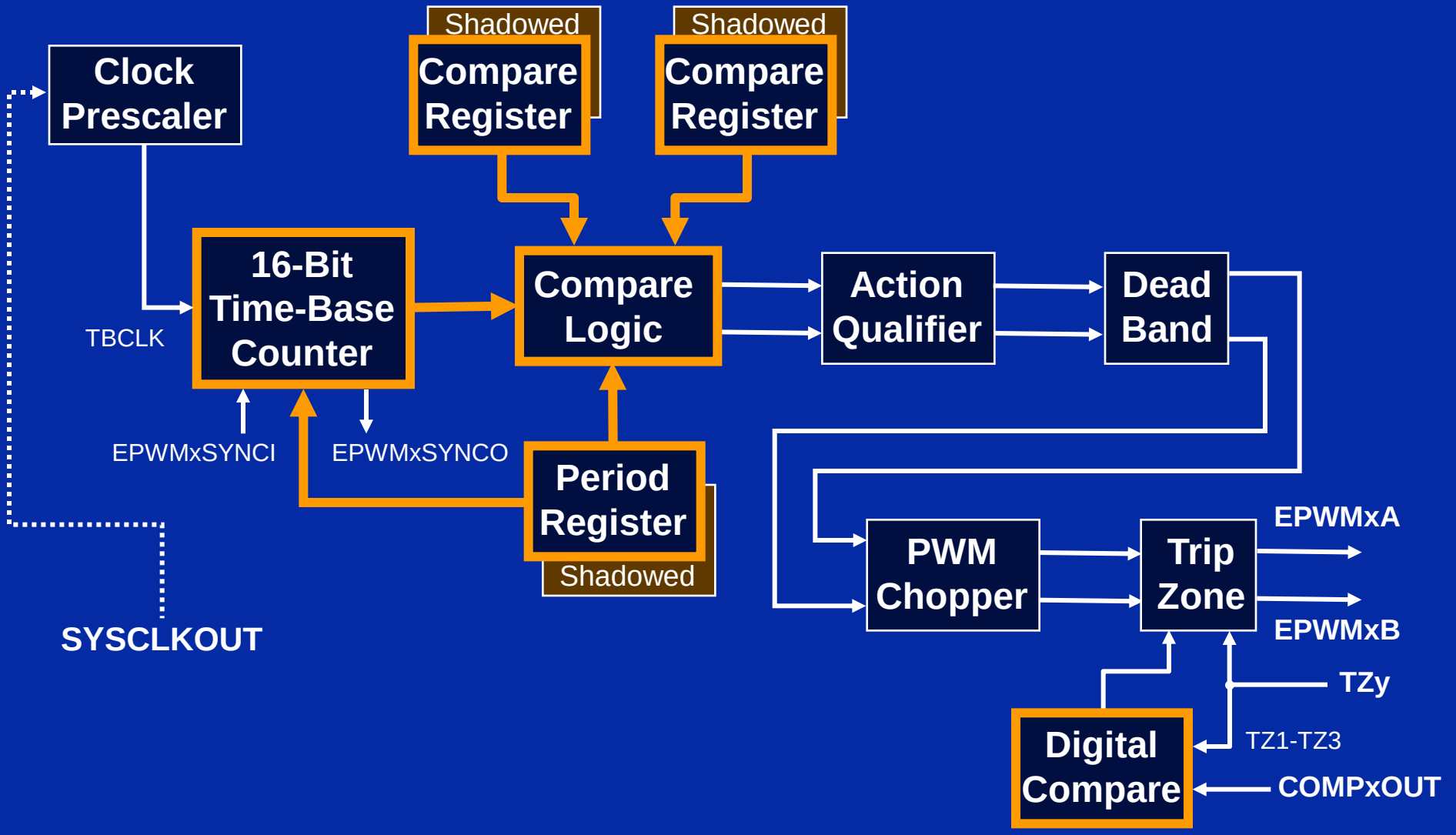
Trip-Zone Features

- Trip-Zone has a fast, clock independent logic path to high-impedance the EPWMxA/B output pins
- Interrupt latency may not protect hardware when responding to over current conditions or short-circuits through ISR software
- Supports: #1) one-shot trip for major short circuits or over current conditions

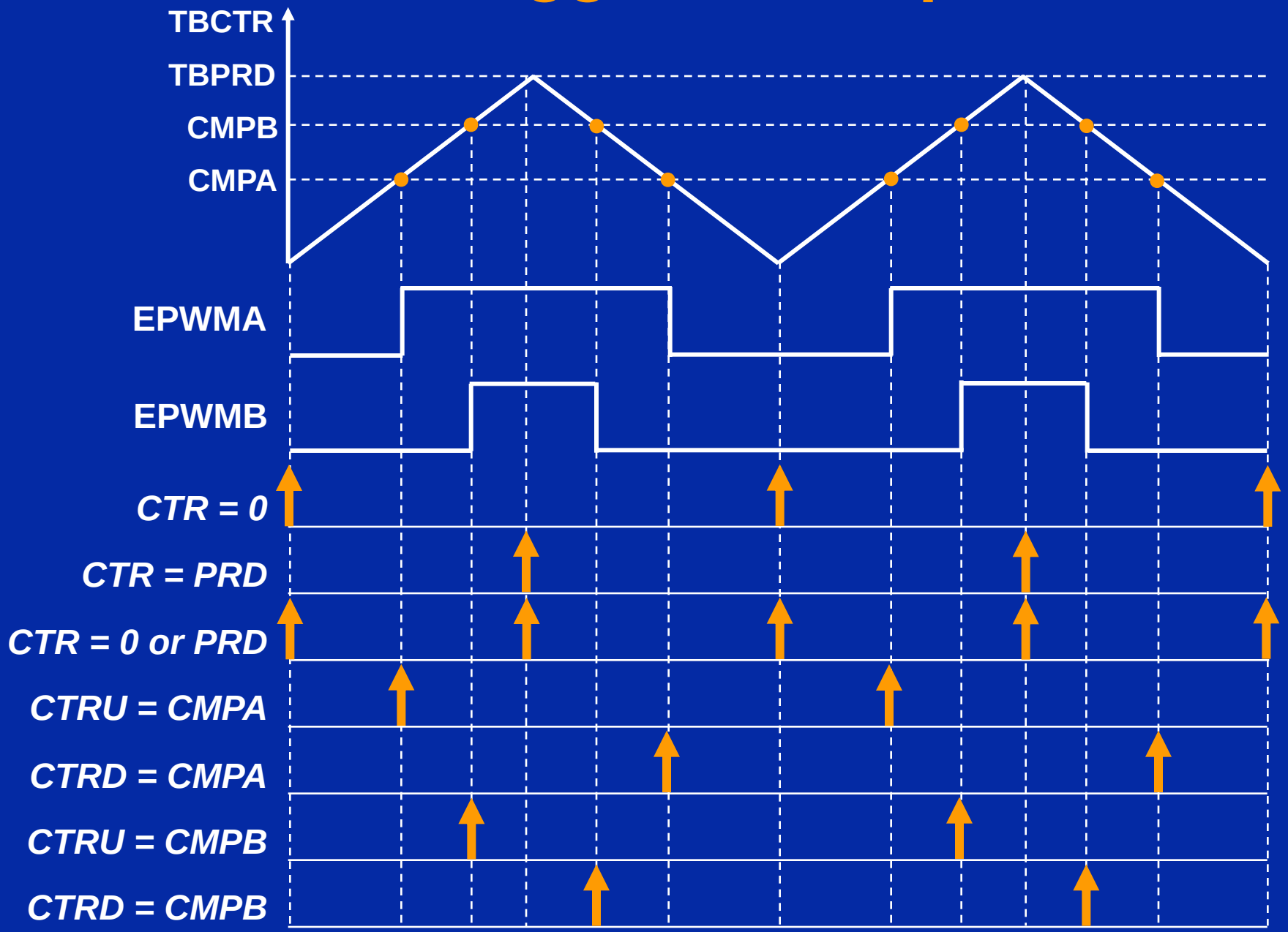
  #2) cycle-by-cycle trip for current limiting operation



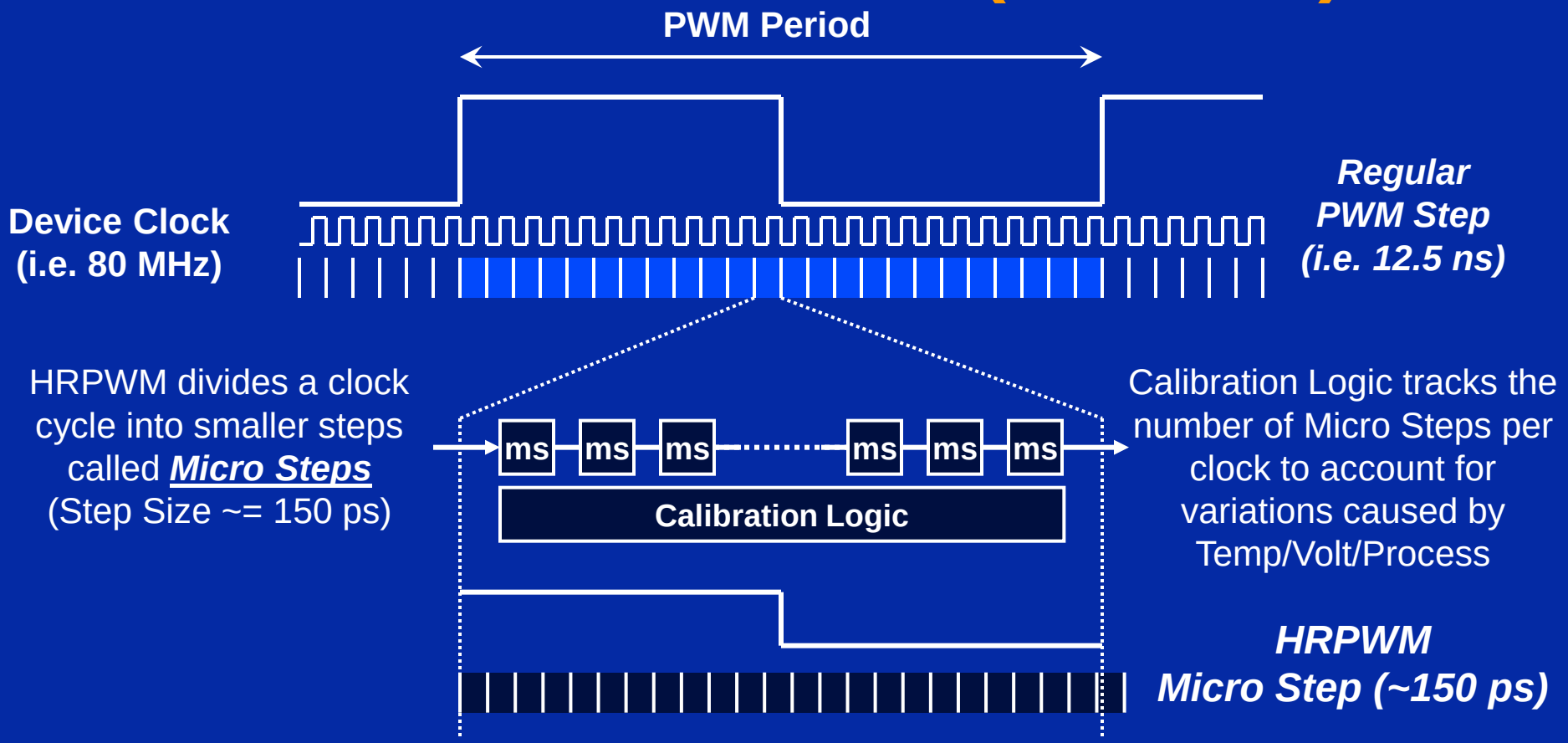
ePWM Event-Trigger Sub-Module



ePWM Event-Trigger Interrupts and SOC



Hi-Resolution PWM (HRPWM)



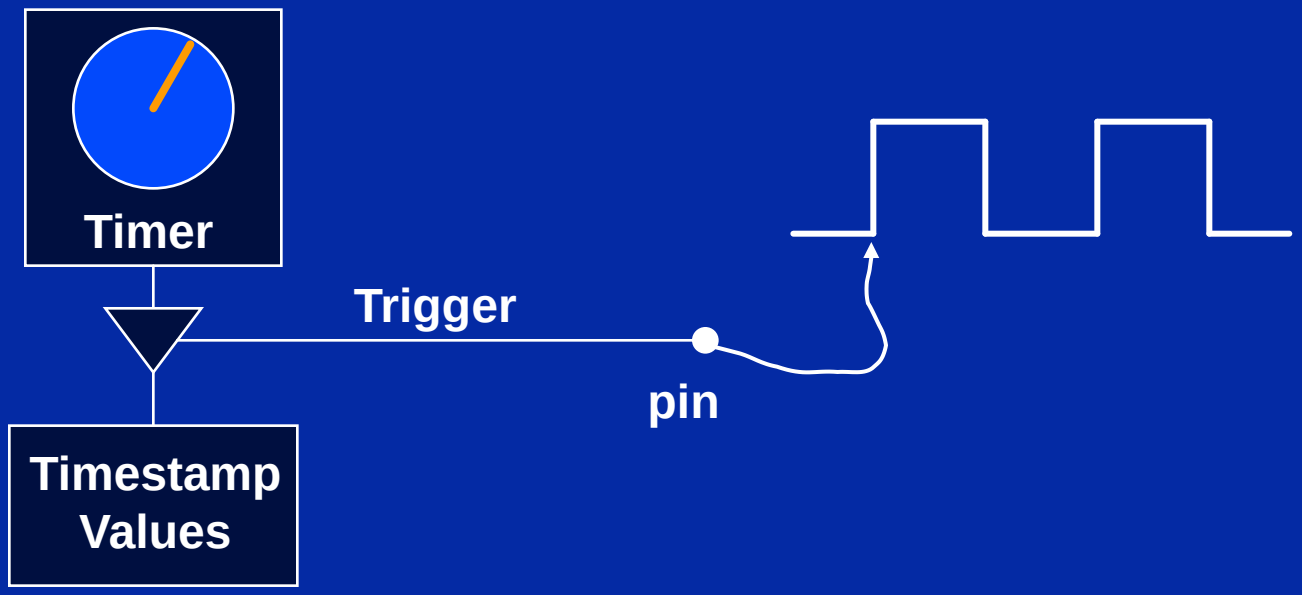
- Significantly increases the resolution of conventionally derived digital PWM
- Uses 8-bit extensions to Compare registers (CMPxHR), Period register (TBPRDHR) and Phase register (TBPHSHR) for edge positioning control
- Typically used when PWM resolution falls below ~9-10 bits which occurs at frequencies greater than ~160 kHz (with system clock of 80 MHz)
- Not all ePWM outputs support HRPWM feature (see device datasheet)

ePWM Control Registers (file: EPwm.c)

- TBCTL (Time-Base Control)
  - counter mode (up, down, up & down, stop); clock prescale; period shadow load; phase enable/direction; sync select
- CMPCTL (Compare Control)
  - compare load mode; operating mode (shadow / immediate)
- AQCTLA/B (Action Qualifier Control Output A/B)
  - action on up/down CTR = CMPA/B, PRD, 0 (nothing/set/clear/toggle)
- DBCTL (Dead-Band Control)
  - in/out-mode (disable / delay PWMxA/B); polarity select
- PCCTL (PWM-Chopper Control)
  - enable / disable; chopper CLK freq. & duty cycle; 1-shot pulse width
- DCTRIPSEL (Digital Compare Trip Select)
  - Digital compare A/B high/low input source select
- TZCTL (Trip-Zone Control)
  - enable /disable; action (force high / low / high-Z /nothing)
- ETSEL (Event-Trigger Selection)
  - interrupt & SOCA/B enable / disable; interrupt & SOCA/B select

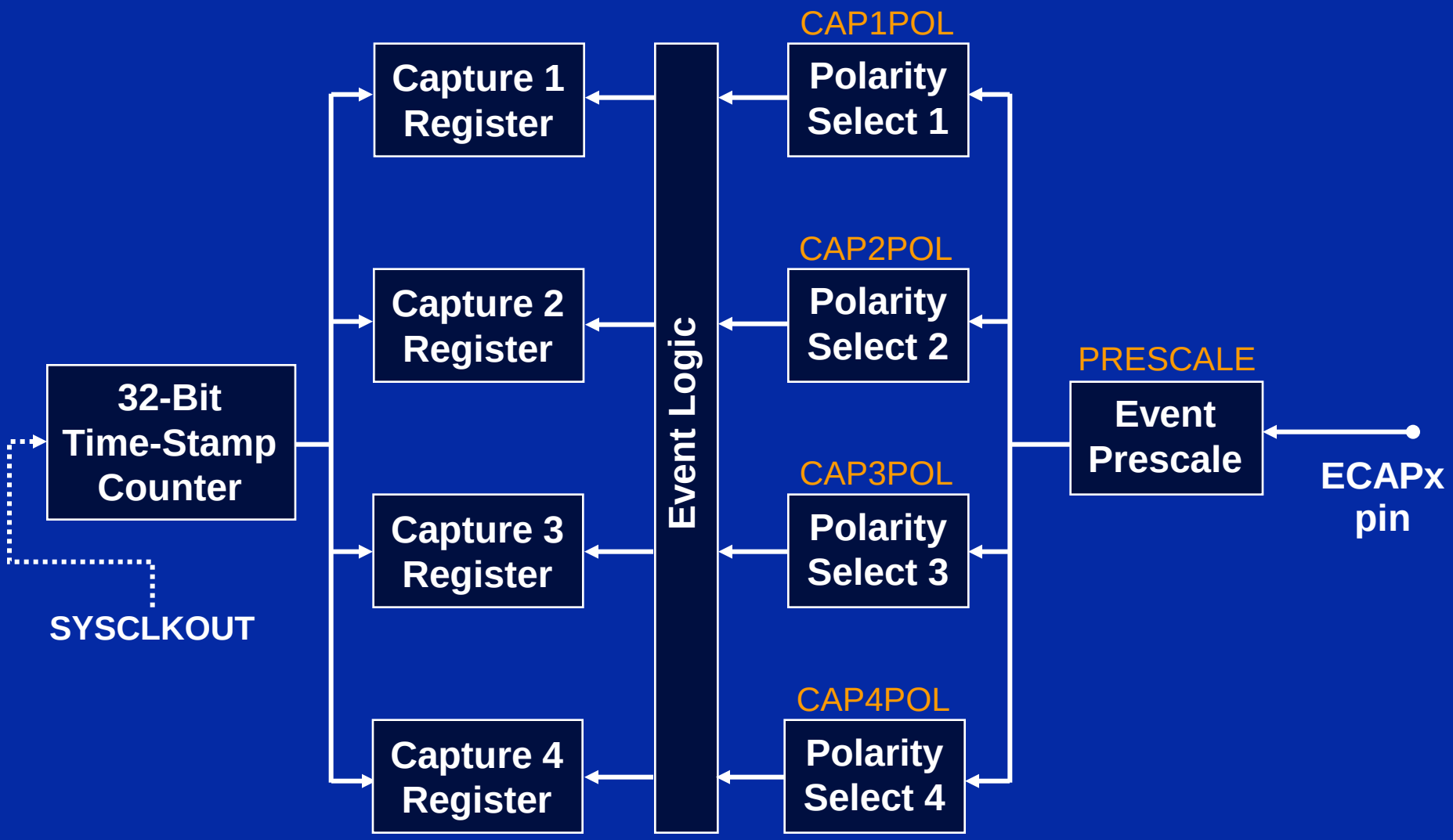
Note: refer to the reference guide for a complete listing of registers

Capture Module (eCAP)

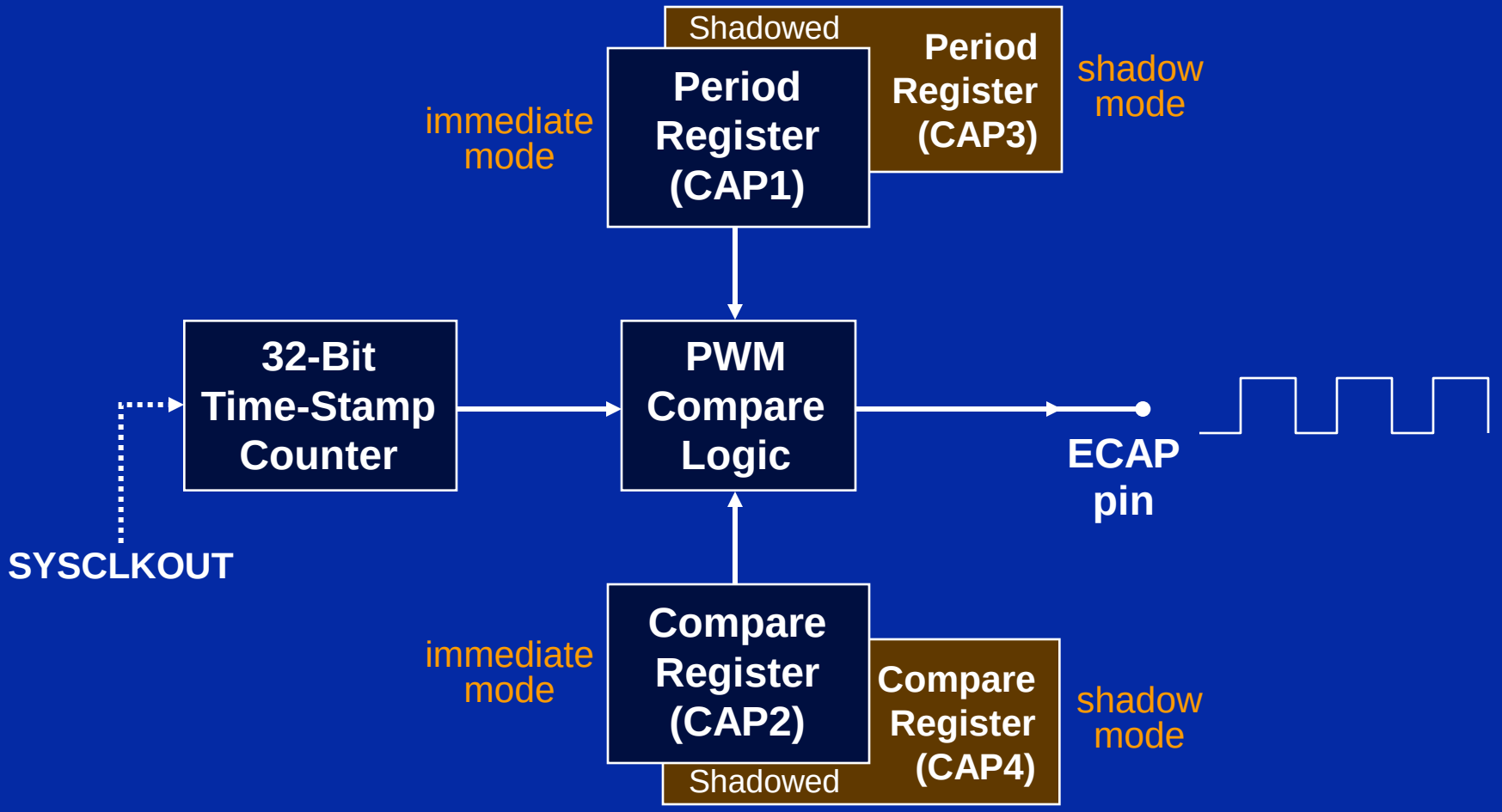


- The eCAP module timestamps transitions on a capture input pin

eCAP Module Block Diagram – Capture Mode

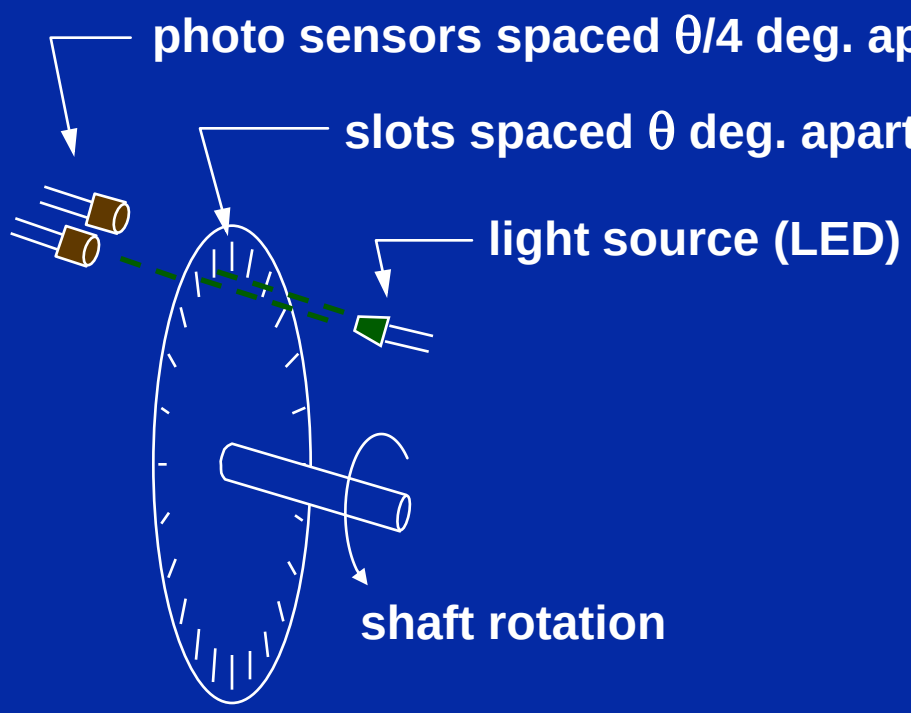


eCAP Module Block Diagram – APWM Mode

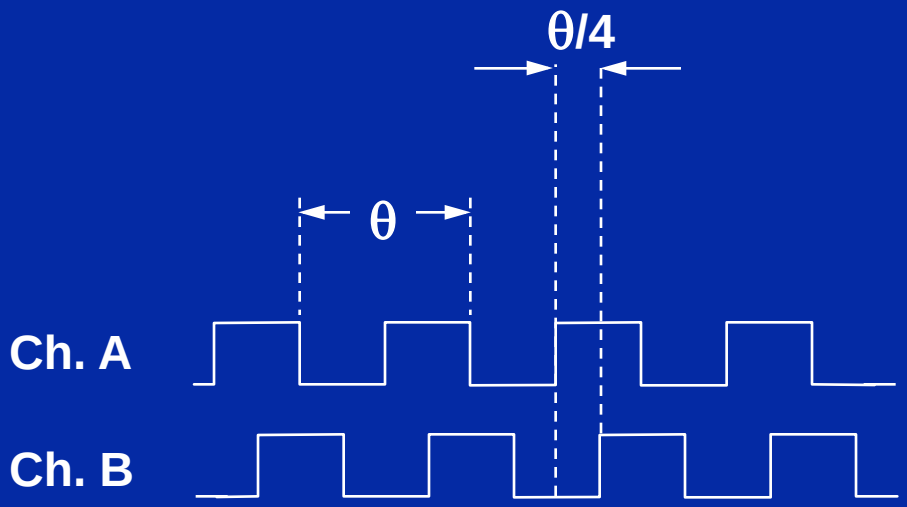


What is an Incremental Quadrature Encoder?

A digital (angular) position sensor



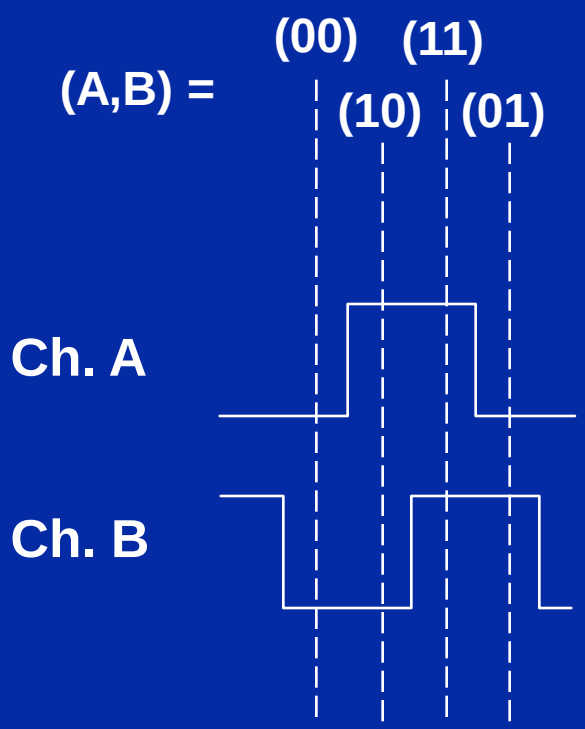
Incremental Optical Encoder

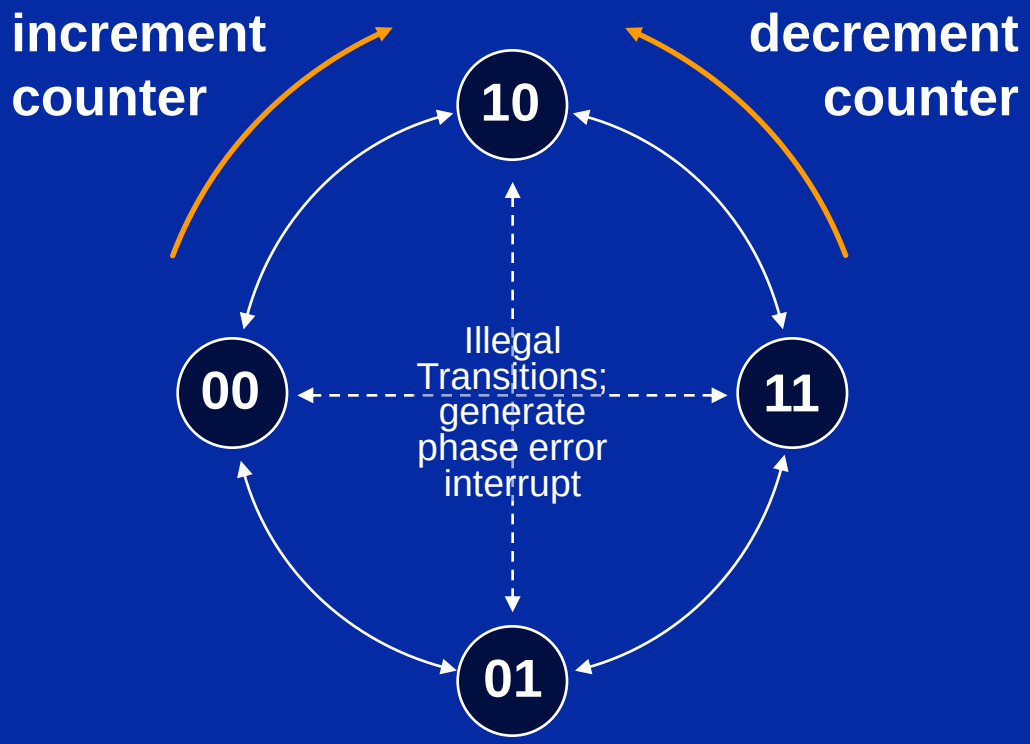


Quadrature Output from Photo Sensors

How is Position Determined from Quadrature Signals?

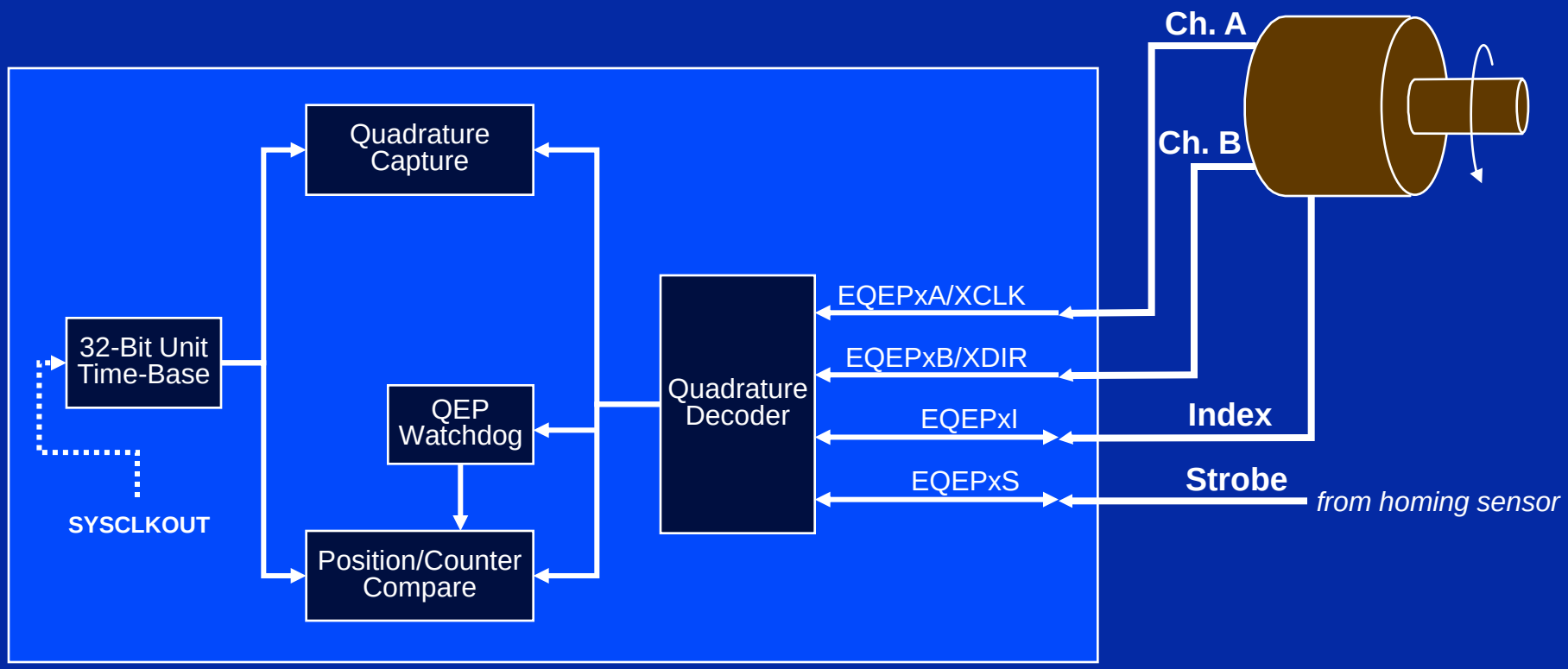
Position resolution is θ/4 degrees



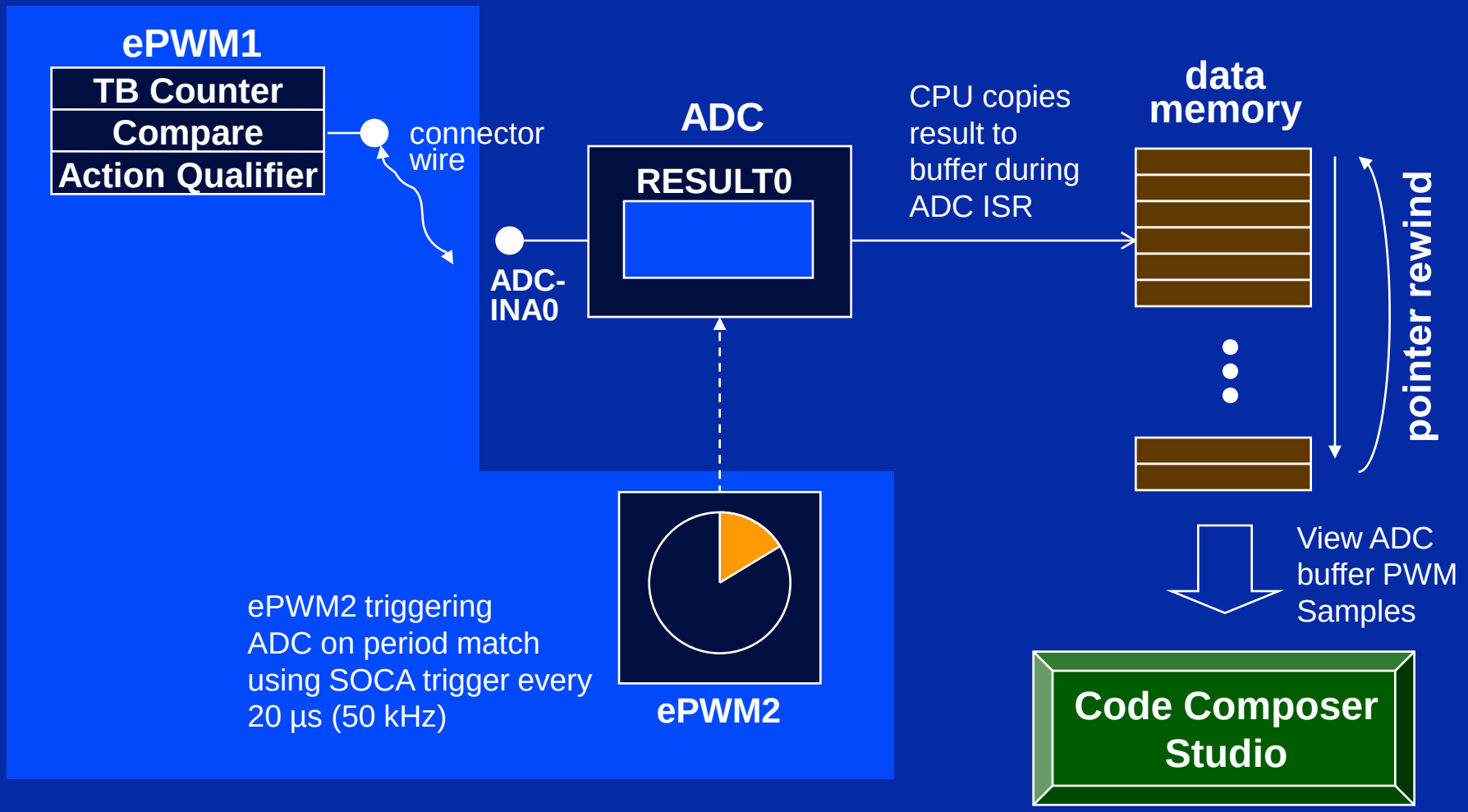


Quadrature Decoder State Machine

eQEP Module Connections



Lab 2: Control Peripherals

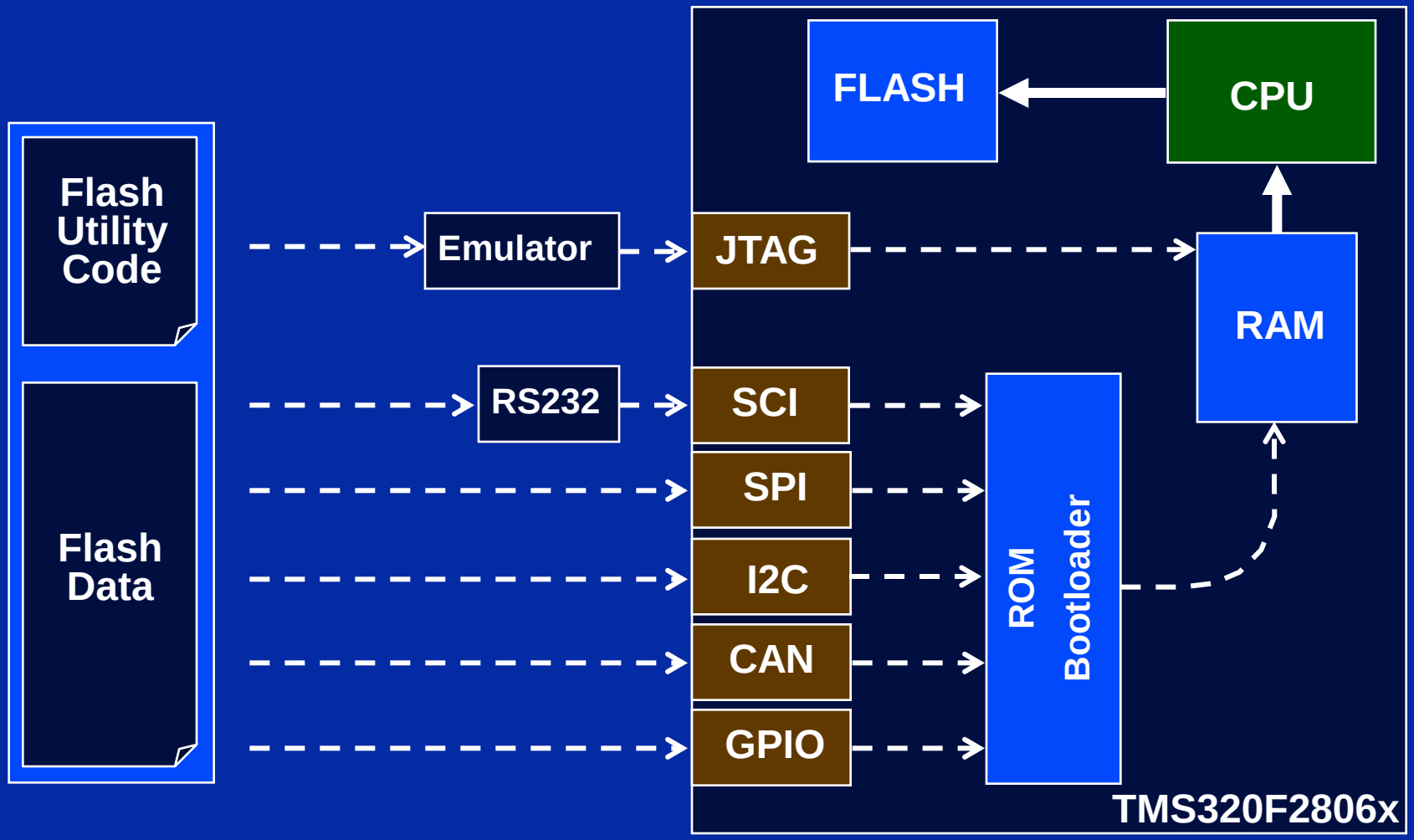


C2000 MCU 1-Day Workshop Outline

- **Workshop Introduction**
- **Architecture Overview**
- **Programming Development Environment**
  - *Lab: Linker command file*
- **Peripheral Register Header Files**
- **Reset, Interrupts and System Initialization**
  - *Lab: Watchdog and interrupts*
- **Control Peripherals**
  - *Lab: Generate and graph a PWM waveform*
- **Flash Programming**
  - *Lab: Run the code from flash memory*
- **The Next Step…**

Flash Programming Basics

- The DSP CPU itself performs the flash programming
- The CPU executes Flash utility code from RAM that reads the Flash data and writes it into the Flash
- We need to get the Flash utility code and the Flash data into RAM



Flash Programming Basics

- Sequence of steps for Flash programming:

Algorithm	Function
1. Erase	- Set all bits to zero, then to one
2. Program	- Program selected bits with zero
3. Verify	- Verify flash contents

- Minimum **Erase** size is a sector (8Kw or 16Kw)
- Minimum **Program** size is a bit!
- Important not to lose power during erase step: If CSM passwords happen to be all zeros, the CSM will be permanently locked!
- Chance of this happening is quite small! (Erase step is performed sector by sector)

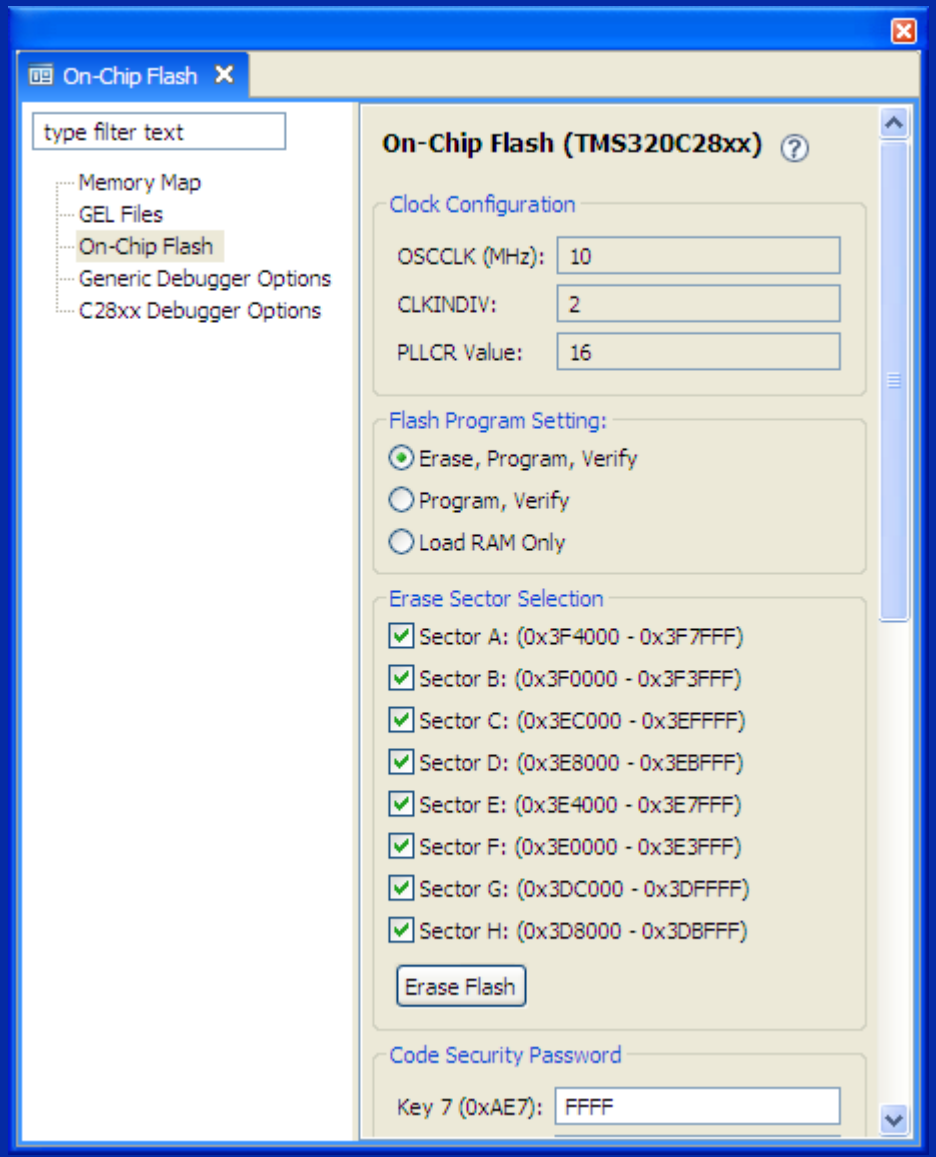
Flash Programming Utilities

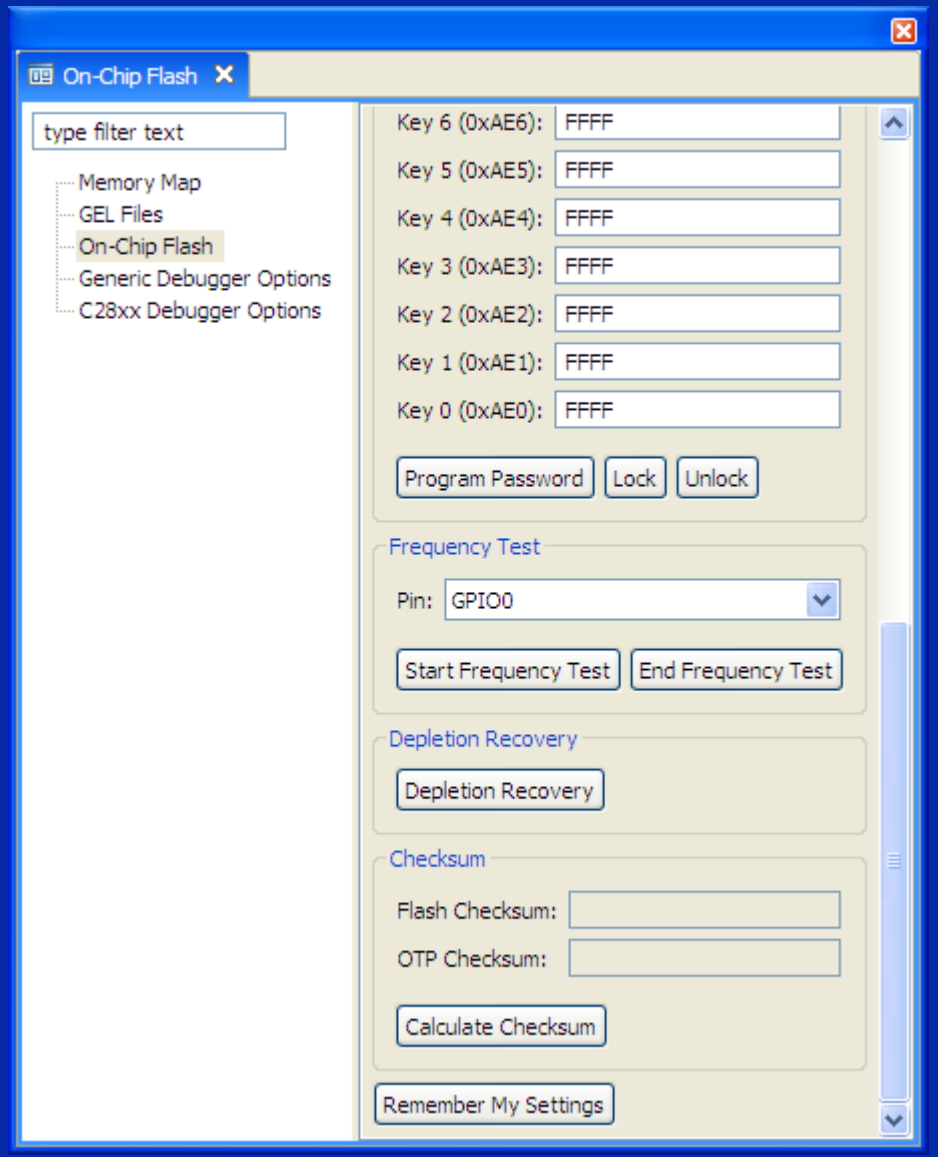
- **JTAG Emulator Based**
  - **Code Composer Studio on-chip Flash programmer**
  - **BlackHawk Flash utilities (requires Blackhawk emulator)**
  - **Elprotronic FlashPro2000**
  - **Spectrum Digital SDFlash JTAG (requires SD emulator)**
  - **Signum System Flash utilities (requires Signum emulator)**
- **SCI Serial Port Bootloader Based**
  - **Code-Skin (http://www.code-skin.com)**
  - **Elprotronic FlashPro2000**
- **Production Test/Programming Equipment Based**
  - **BP Micro programmer**
  - **Data I/O programmer**
- **Build your own custom utility**
  - **Can use any of the ROM bootloader methods**
  - **Can embed flash programming into your application**
  - **Flash API algorithms provided by TI**

* TI web has links to all utilities (http://www.ti.com/c2000)

CCS On-Chip Flash Programmer

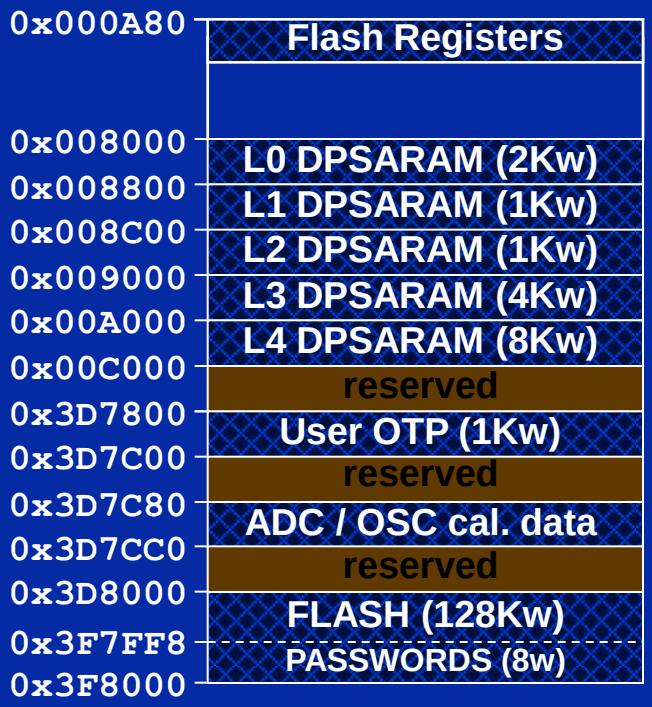
- On-Chip Flash programmer is integrated into the CCS debugger





Code Security Module (CSM)

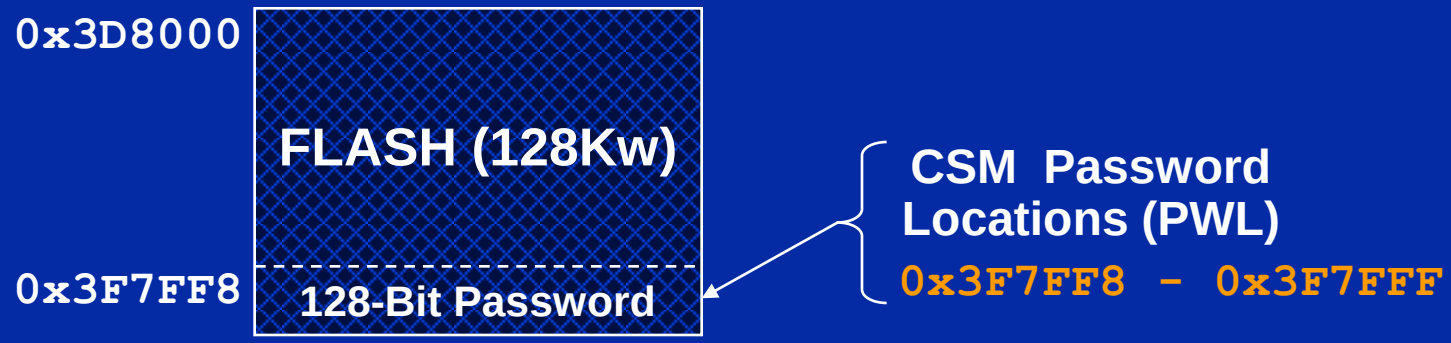
- Access to the following on-chip memory is restricted:



- Data reads and writes from restricted memory are only allowed for code running from restricted memory
- All other data read/write accesses are blocked:

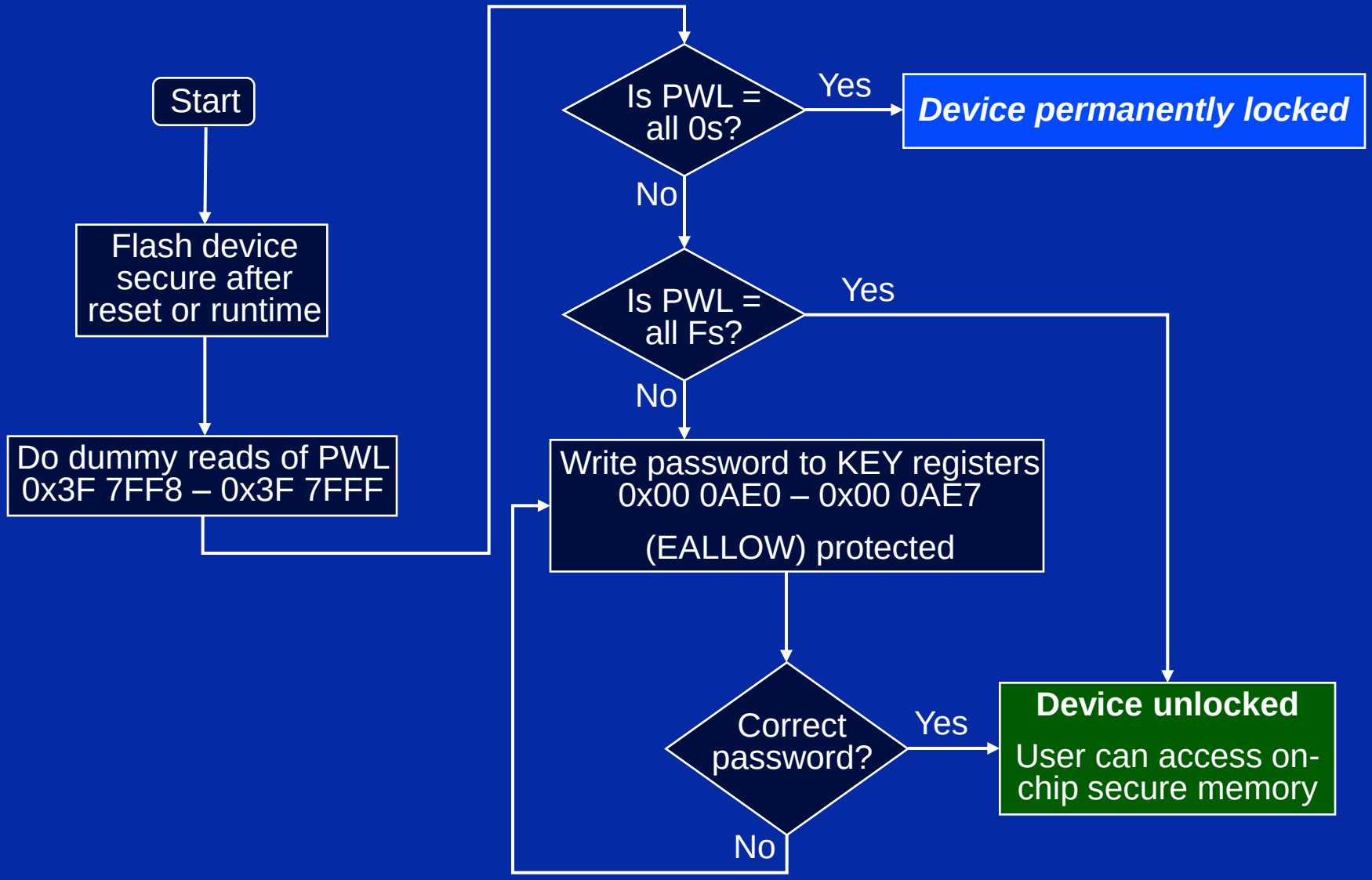
  JTAG emulator/debugger, ROM bootloader, code running in external memory or unrestricted internal memory

CSM Password

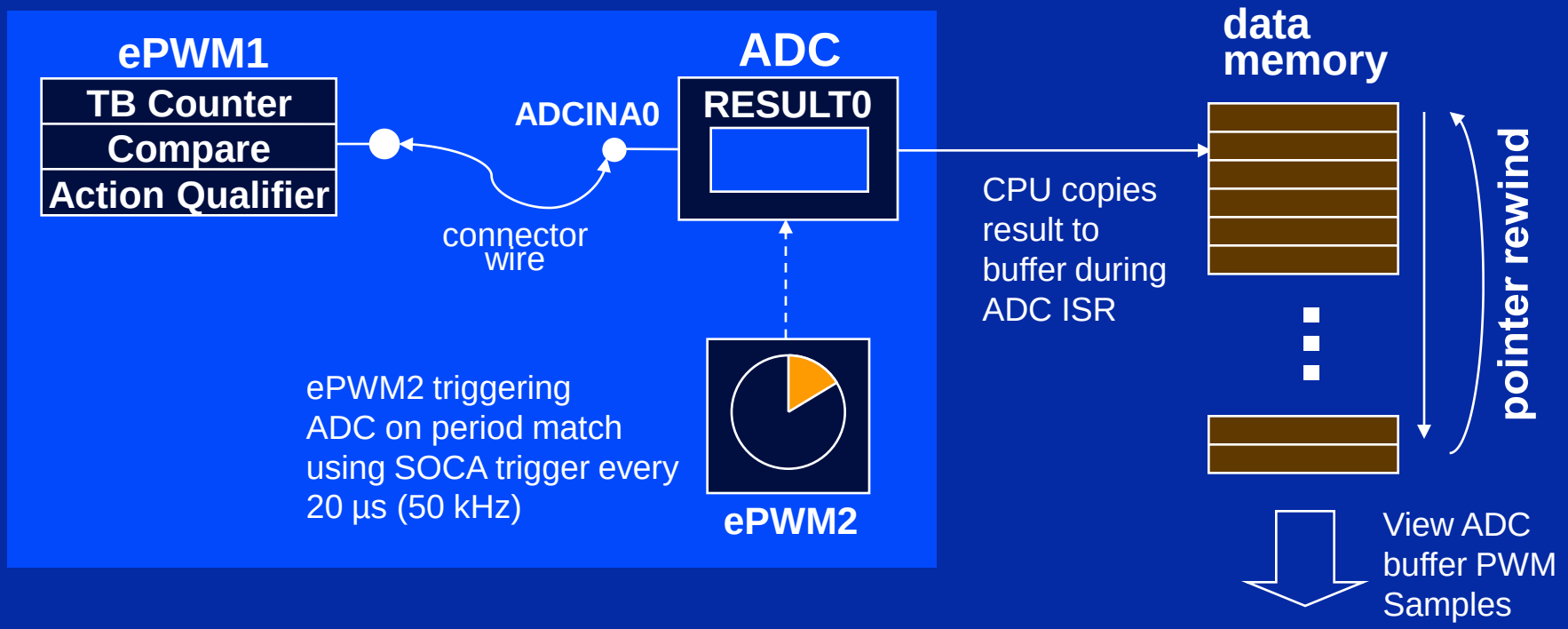


- 128-bit user defined password is stored in Flash
- 128-bit KEY registers are used to lock and unlock the device
  - Mapped in memory space 0x00 0AE0 – 0x00 0AE7
  - Registers “EALLOW” protected

CSM Password Match Flow



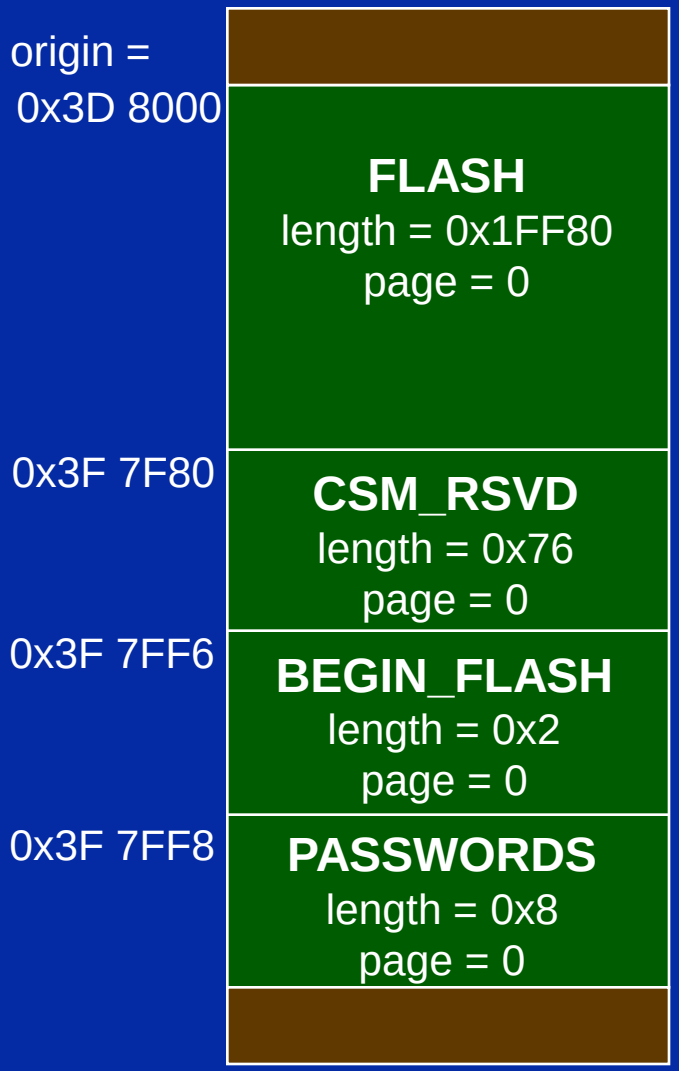
Lab 3: Programming the Flash



Objective:

- **Program system into Flash Memory**
- **Learn use of CCS Flash Programmer**
- ***DO NOT PROGRAM PASSWORDS***

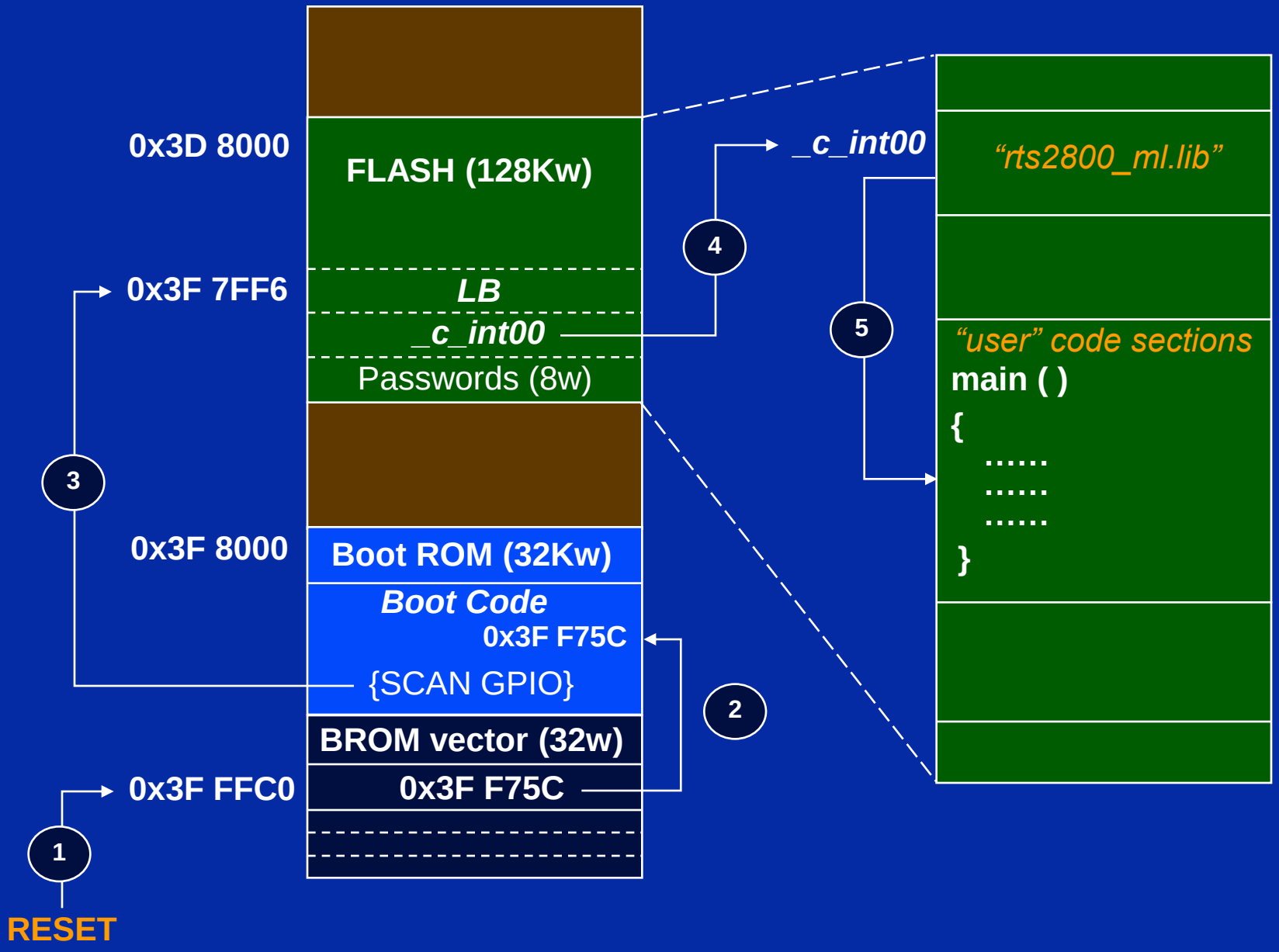
Flash Memory Section Blocks



Lab_4.cmd

```
SECTIONS
{
   codestart    :>  BEGIN_FLASH,  PAGE = 0
   passwords  :>  PASSWORDS,   PAGE = 0
   csm_rsvd    :>  CSM_RSVD,     PAGE = 0
}
```

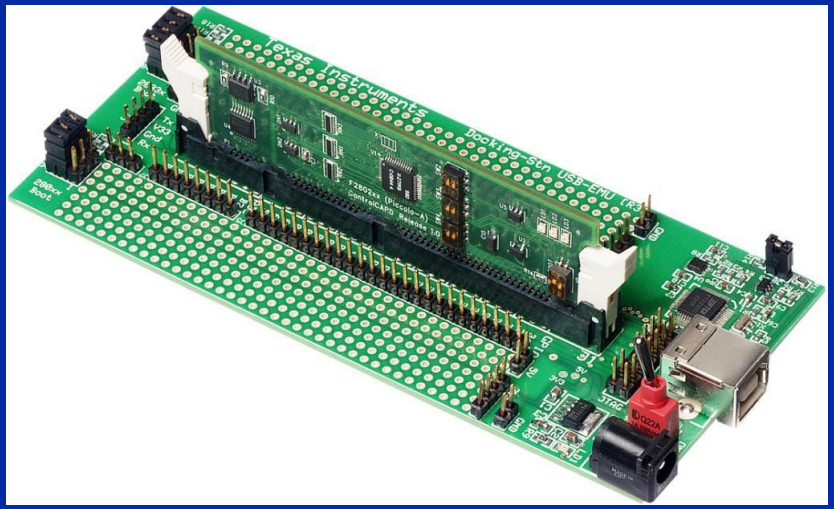
Startup Sequence from Flash Memory



C2000 MCU 1-Day Workshop Outline

- Workshop Introduction
- Architecture Overview
- Programming Development Environment
  - *Lab: Linker command file*
- Peripheral Register Header Files
- Reset, Interrupts and System Initialization
  - *Lab: Watchdog and interrupts*
- Control Peripherals
  - *Lab: Generate and graph a PWM waveform*
- Flash Programming
  - *Lab: Run the code from flash memory*
- **The Next Step…**

C2000 MCU Multi-day Training Course



In-depth hands-on TMS320F28069 Design and Peripheral Training

TMS320F2806x Workshop Outline

- Architectural Overview
- Programming Development Environment
- Peripheral Register Header Files
- Reset and Interrupts
- System Initialization
- Analog-to-Digital Converter
- Control Peripherals
- Numerical Concepts and Iqmath
- Direct Memory Access (DMA)
- Control Law Accelerator (CLA)
- Viterbi, Complex Math, CRC Unit (VCU)
- System Design
- Communications
- DSP/BIOS
- Support Resources

controlSUITE™

Texas Instruments controlSUITE

GUI exclusively powered by CROSSHAIRS EMBEDDED
crosshairsembedded.com

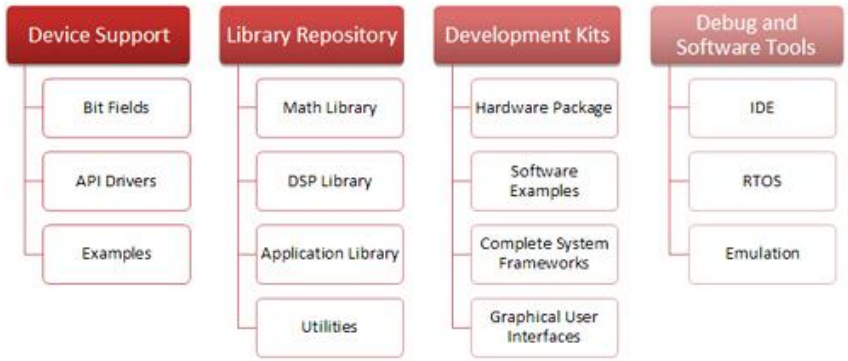
controlSUITE™

- controlSUITE
- Devices
- Kits
- Libraries
- Update
- Application Notes
- Training and Support
- Datasheets and Guides
- Code Composer Studio IDE
- Search results

controlSUITE™

Comprehensive. Intuitive. Optimized. Real world software for real-time control.

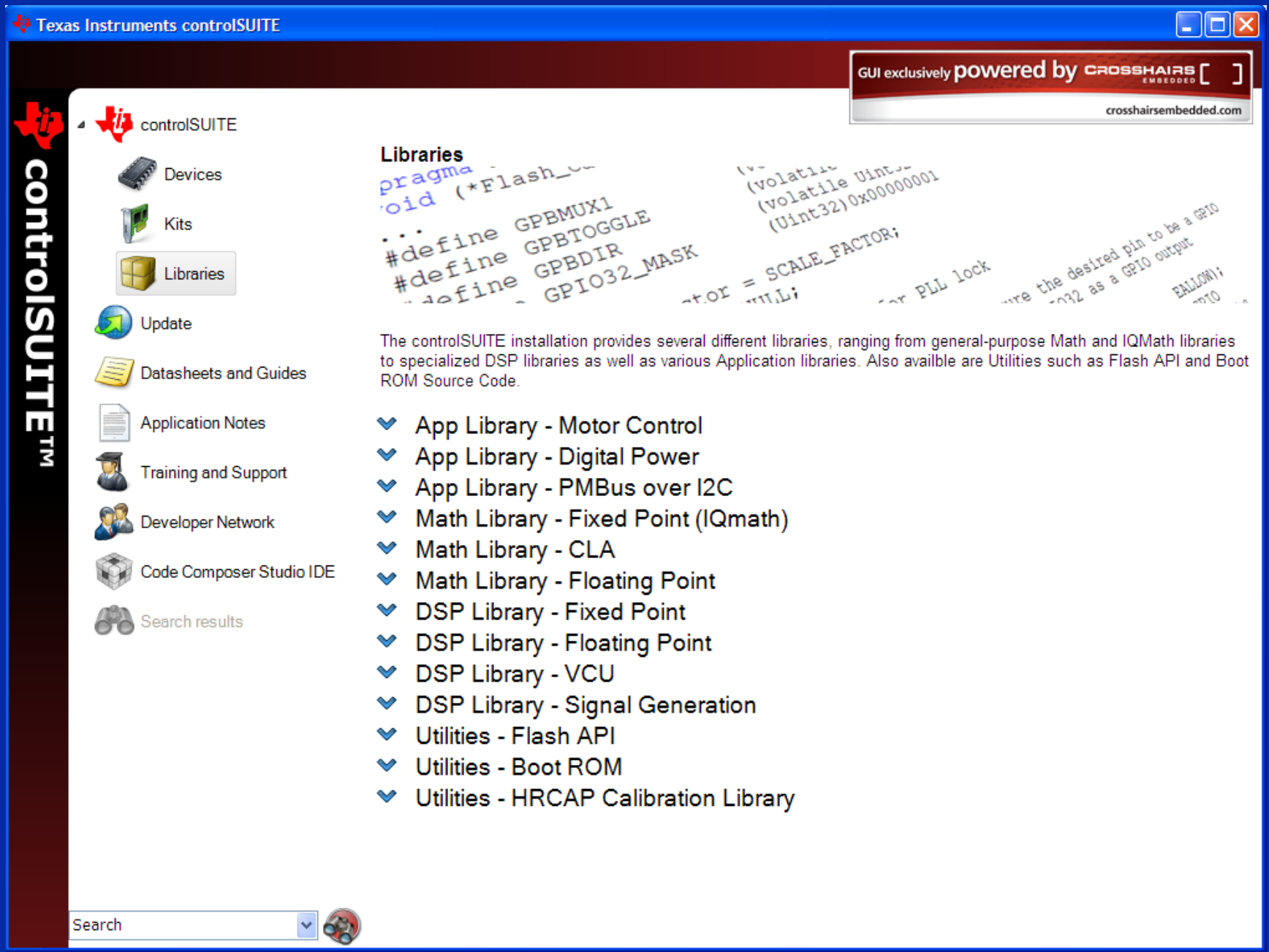
controlSUITE™ for C2000™ microcontrollers is a cohesive set of software infrastructure and software tools designed to minimize software development time. From device-specific drivers and support software to complete system examples in sophisticated system applications, controlSUITE™ provides libraries and examples at every stage of development and evaluation. Go beyond simple code snippits - jump start your real-time system with real-world software.



Search

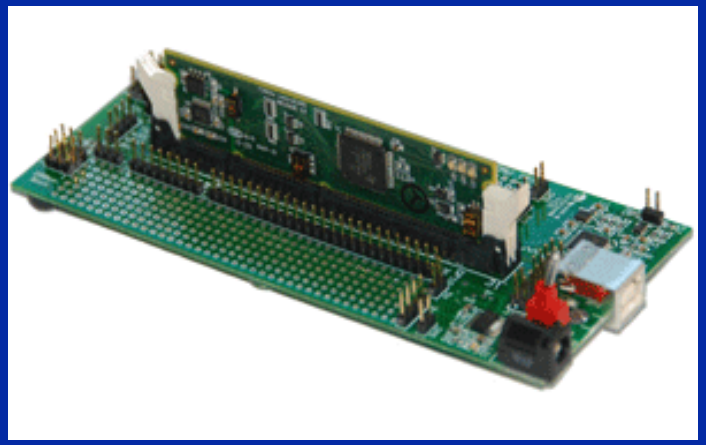
Software

controlSUITE™ Libraries



C2000 Experimenter's Kits

F28069, F28035, F28027, F28335, F2808

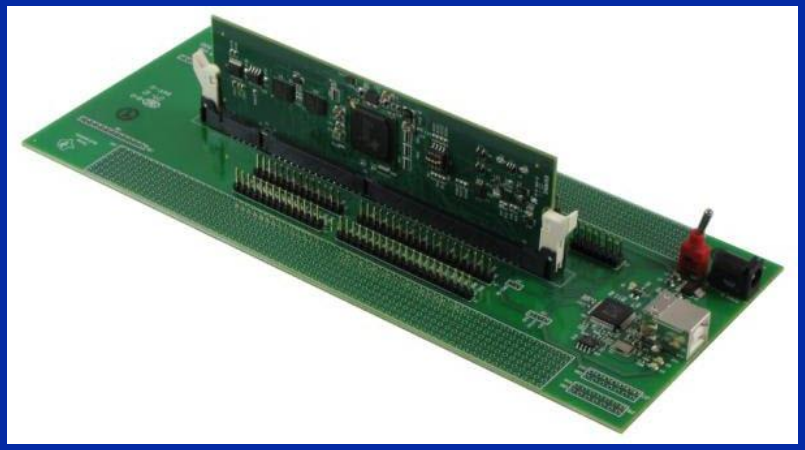


- Part Number:
  - TMDXDOCK28069
  - TMDSDOCK28035
  - TMDSDOCK28027
  - TMDSDOCK28335
  - TMDSDOCK2808

- Experimenter Kits include
  - F28069, F28035, F28027, F28335 or F2808 controlCARD
  - USB docking station
  - C2000 Applications Software CD with example code and full hardware details
  - Code Composer Studio v4
- Docking station features
  - Access to controlCARD signals
  - Breadboard areas
  - Onboard USB JTAG Emulation
    - *JTAG emulator not required*
- Available through TI authorized distributors and the TI eStore

C2834x Experimenter's Kits

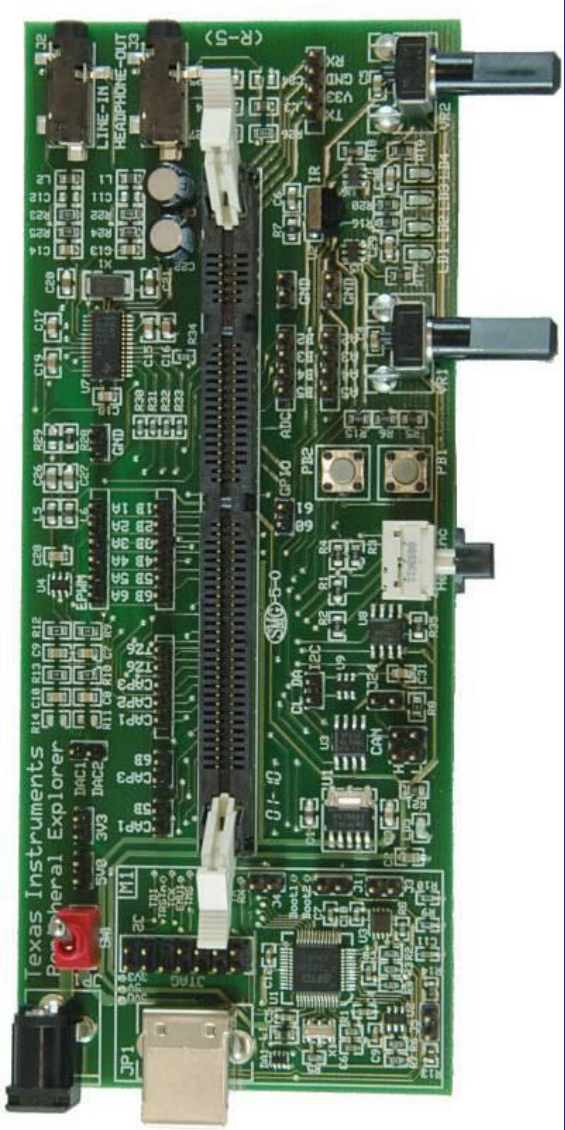
C28343, C28346



- Part Number:
  - TMDXDOCK28343
  - TMDSDOCK28346-168

- Experimenter Kits include
  - C2834x controlCARD
  - Docking station
  - C2000 Applications Software CD with example code and full hardware details
  - Code Composer Studio v4
  - 5V power supply
- Docking station features
  - Access to controlCARD signals
  - Breadboard areas
  - *JTAG emulator required – sold separately*
- Available through TI authorized distributors and the TI eStore

F28335 Peripheral Explorer Kit



TMDSPREX28335

- Experimenter Kit includes
  - F28335 controlCARD
  - Peripheral Explorer baseboard
  - C2000 Applications Software CD with example code and full hardware details
  - Code Composer Studio v4
- Peripheral Explorer features
  - ADC input variable resistors
  - GPIO hex encoder & push buttons
  - eCAP infrared sensor
  - GPIO LEDs, I2C & CAN connection
  - Analog I/O (AIC+McBSP)
- Onboard USB JTAG Emulation
  - *JTAG emulator not required*
- Available through TI authorized distributors and the TI eStore

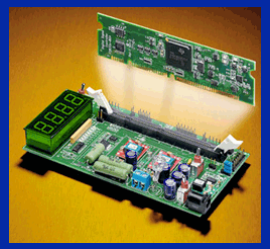
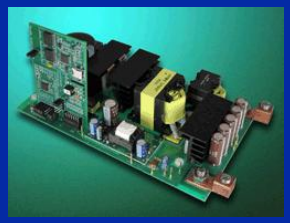
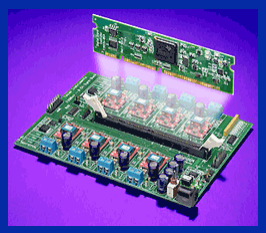
C2000 controlSTICK Evaluation Tool

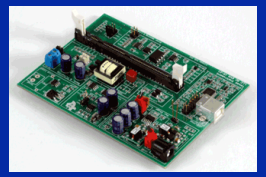
F28069, F28027

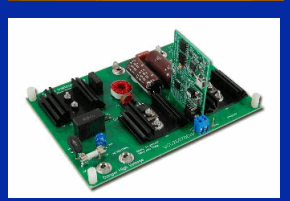
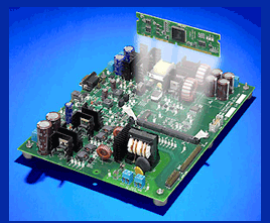
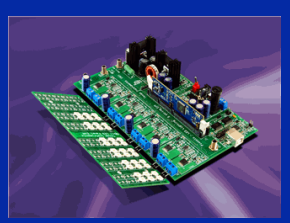
- Part Number:
  - TMDX28069USB
  - TMDS28027USB

- Low-cost USB evaluation tool
- Onboard JTAG Emulation
  - *JTAG emulator not required*
- Access to controlSTICK signals
- C2000 Applications Software CD with example code and full hardware details
- Code Composer Studio v4
- Available through TI authorized distributors and the TI eStore

C2000 controlCARD Application Kits

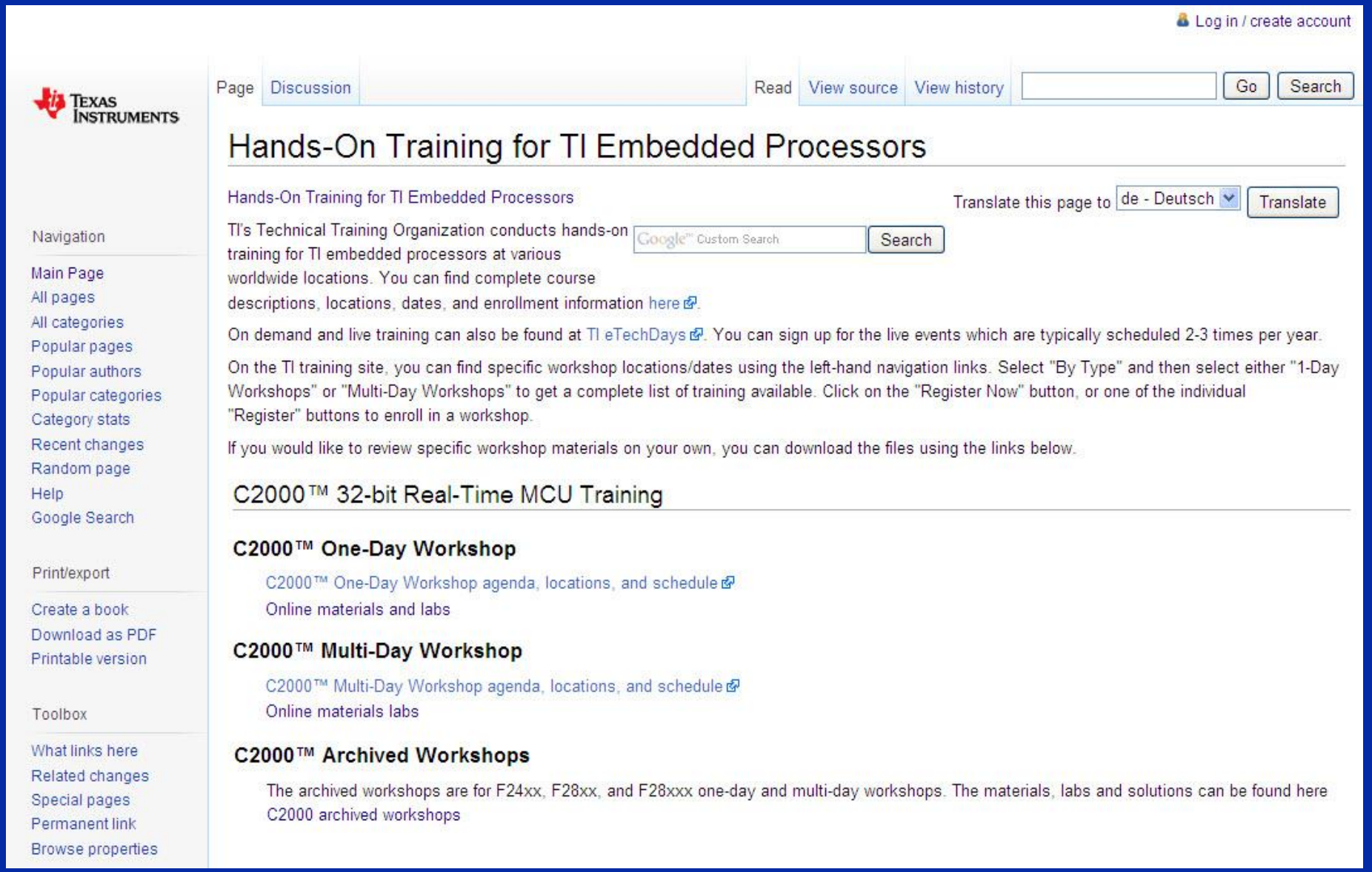






- Developer's Kit for – *Motor Control, PFC, High Voltage, Digital Power, Renewable Energy, LED Lighting, etc.*
- Kits includes
  - controlCARD and application specific baseboard
  - Full version of Code Composer Studio v4
- Software download includes
  - Complete schematics, BOM, gerber files, and source code for board and all software
  - Quickstart demonstration GUI for quick and easy access to all board features
  - Fully documented software specific to each kit and application
- See www.ti.com/c2000 for other kits and more details
- Available through TI authorized distributors and the TI eStore

C2000 Workshop Download Wiki



http://processors.wiki.ti.com/index.php/Training

For More Information . . .

- USA – Product Information Center (PIC)
  - Phone: 800-477-8924 or 972-644-5580
  - E-mail: support@ti.com
- TI E2E Community (videos, forums, blogs)
  - http://e2e.ti.com
- Embedded Processor Wiki
  - http://processors.wiki.ti.com
- TI Training
  - http://www.ti.com/training
- TI eStore
  - http://estore.ti.com
- TI website
  - http://www.ti.com

TEXAS INSTRUMENTS

Technical Training Organization

www.ti.com/training