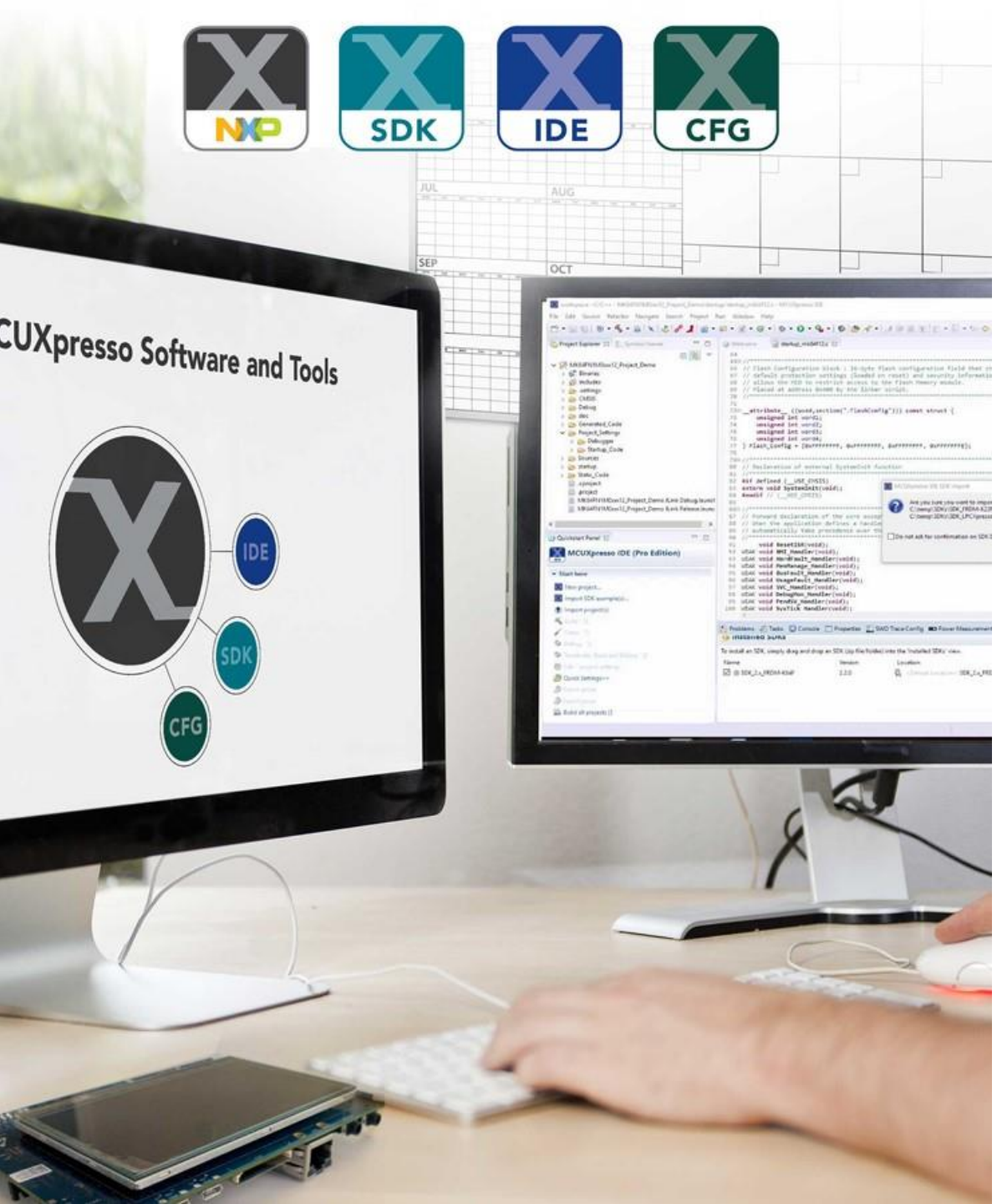
CREATIVE WAYS TO LEVERAGE THE LPC804 MCU'S INTEGRATED PROGRAMMABLE LOGIC FEATURE

JUNE 2018

PUBLIC

SECURE CONNECTIONS FOR A SMARTER WORLD



Agenda

- LPC800 Series Introduction/Recap
- MCUXpresso SW & Tools Overview
- The LPC804 MCU Programmable Logic Unit (PLU)
- The PLU Design Tool
- Application Examples
- Conclusion

LPC Microcontrollers
Broad Market Leader

Architecting Scalable MCU Families with Flexible Integration
Enabling Fast Time & Platform Re-use

1. **Innovative Arm® MCU Portfolio**
2. **Ecosystem & Partners**
3. **Supply, Longevity & Quality**
4. **Local Support Network**
5. **Extensive Software & Tools**

- Accelerating Transition from 8-bit to **Entry-level Arm® Cortex®-M0+ based MCUs**
- **Low Power, High Performance MCUs** for Energy Conscious Application

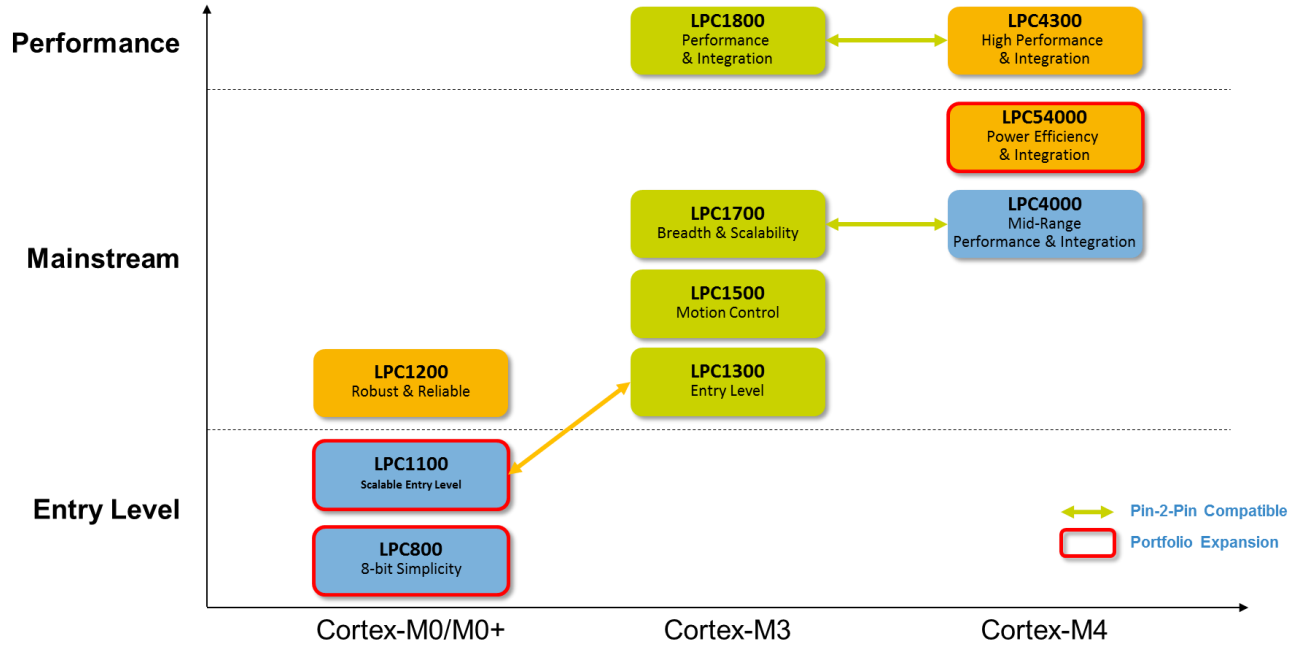
LPC 32-bit Microcontrollers for the Mass Market

Over 1B units shipped	>400 part numbers	Thriving ecosystem	Complementary professional development suite (HW/SW)

Open Development Environment

- **MCUXpresso IDE with Easy to Use Software Code Bundles**
- **Development, Debug & Expansion Boards**
- **Developer Community**

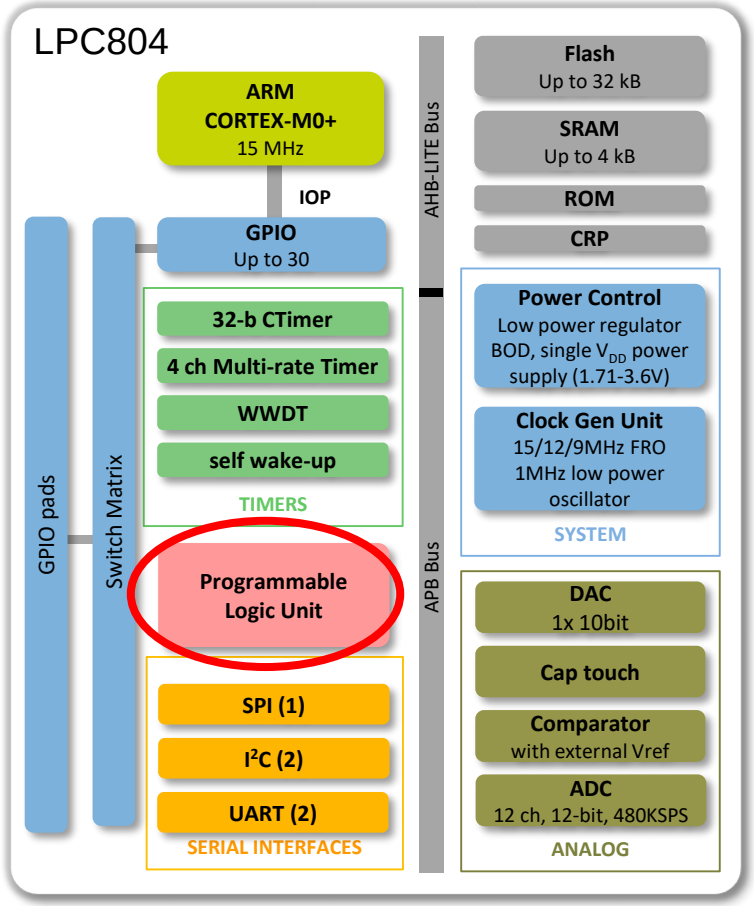
Complete Portfolio of Cortex-M MCUs



Easy Development

Scalable Expansion

LPC804 MCUs: 15MHz Cortex-M0+ with 32KB Flash



System

- 15-MHz Cortex-M0+ Arm core
- Up to 32 kB EEPROM-based Flash, with 64 B page size
- Up to 4 kB RAM

Serial connectivity and GPIOs

- 2 I^2C (Fm), 1 SPI, 2 UART
- Up to 30 GPIOs with pattern matching and level shifting
- Switch matrix for flexible I/O pin assignment

Analog

- 12-bit, 480ksps ADC
- Analog Comparator: 4 input pins
- 1x 10-bit DAC
- **Level shifter option**
- Capacitive Touch

Timers

- 1x 32-bit CTimer
- 4-ch Multi-Rate Timer (MRT)
- Wakeup Timer
- Watchdog Timer
- **Programmable Logic Unit (PLU)**
- For customer defined sequential and combinational logic

Single **power supply**: 1.71 to 3.6V

Temperature range: -40 to +105 °C (ambient)

Packages: TSSOP20, TSSOP24, HVQFN33

LPC800 Series Features

	LPC802	LPC804	LPC8N04	LPC811	LPC812	LPC832	LPC822	LPC834	LPC824	LPC844	LPC845
Frequency	15 MHz		8MHz	30 MHz						30 MHz	
Flash (KB)	16	32	32	8	16	16	16	32	32	64	64
SRAM (KB)	2	4	8	2	4	4	4	4	8	8	16
EEPROM (KB)			4								
FAIM (bits)										256	256
Internal Oscillator	FRO, +/-1%		FRO, +/-1%	IRC, 12MHz +/-1.5%						FRO, +/-1%	
Timer/PWM	32-b Ctimer, MRT		32/16-b CTimer	SCTimer						SCTimer + 32-b CTimer	
USART/SPI/I2C	2/1/1	2/1/2	0 / 1 /1	2/1/1	3/2/1	1/2/1	3/2/4	1/2/1	3/2/4	2/2/2	5/2/4
CRC				1	1	1	1	1	1	1	1
DMA						18-ch				18-ch	
ADC	12-b 400 KSPS					12-b 1.2 MSPS				12-b 1.2 MSPS	
ACMP	1	1		1	1		1		1	1	1
DAC		1x 10-b									2x 10-b
Cap Touch		1									1
PLU		1									
Level Shifter	Yes										
Voltage	1.8-3.6V		1.72-3.6	1.8-3.6V						1.8-3.6V	
Temperature	105C		85C	105C	105C	85C	105C	85C	105C	105C	
Packages	TSSOP16 TSSOP20	TSSOP20 TSSOP24 HVQFN33	HVQFN24	TSSOP16	TSSOP16 XSON16 SO20 TSSOP20	TSSOP20	TSSOP20 HVQFN33	HVQFN33	TSSOP20 HVQFN33	HVQFN33 HVQFN48 LQFP48 LQFP64	HVQFN33 HVQFN48 LQFP48 LQFP64

SOFTWARE & TOOLS OVERVIEW

MCUXpresso suite

MCUXpresso Software and Tools

for Kinetis and LPC microcontrollers

MCUXpresso IDE

Edit, compile, debug and optimize in an intuitive and powerful IDE

MCUXpresso SDK

Runtime software including peripheral drivers, middleware, RTOS, demos and more

LPC800 Series support now available!

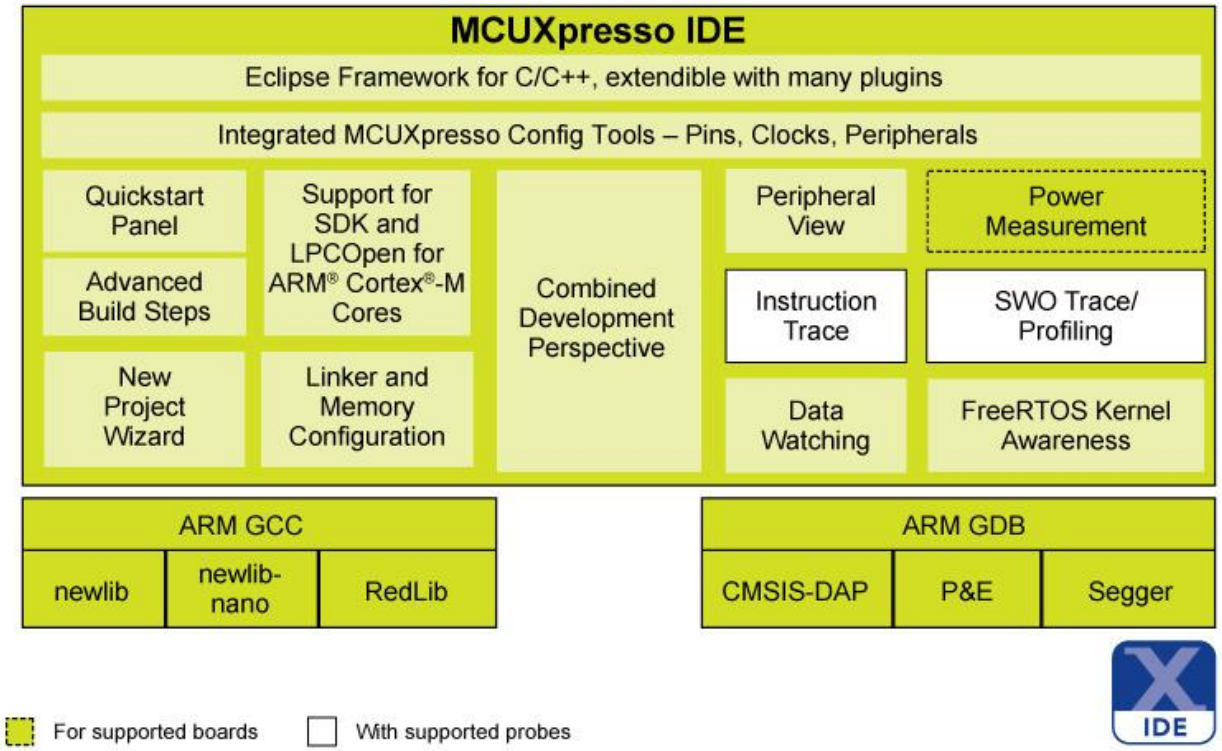
MCUXpresso Config Tools

Online and desktop tool suite for system configuration and optimization

Learn more at: www.nxp.com/mcuxpresso/ide

MCUXpresso IDE

Free Eclipse and GCC-based IDE for C/C++ development on Kinetis and LPC MCUs



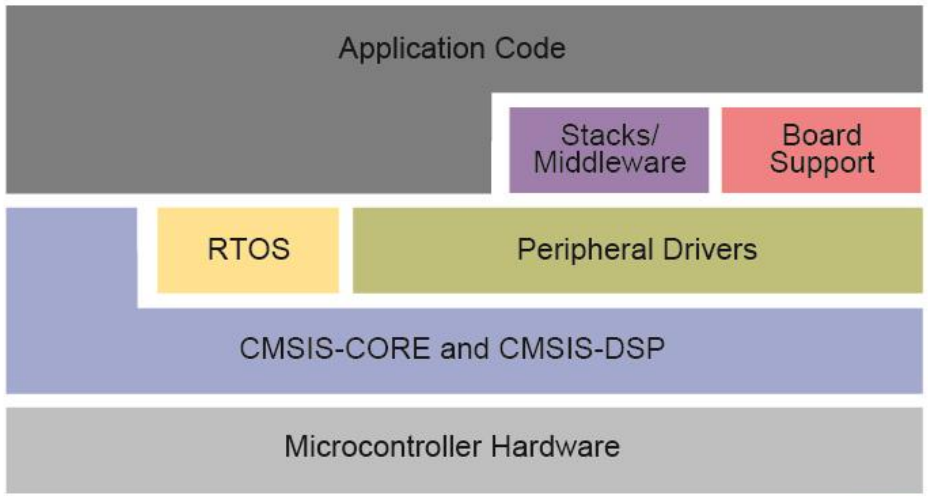
Product Features

- Feature-rich, unlimited code size, optimized for ease-of-use, based on industry standard Eclipse framework for NXP's Kinetis and LPC MCUs, and i.MX RT Crossover Processors
- Application development with Eclipse and GCC-based IDE for advanced editing, compiling and debugging
- Integrated configuration tools for easy project updating
- Supports custom development boards, Freedom, Tower and LPCXpresso boards with debug probes from NXP, P&E and Segger
- Non-intrusive, real-time data watch and printf via SWD
- No activation needed and free community based support

Learn more at: www.nxp.com/mcuxpresso/sdk

MCUXpresso SDK

The software framework and reference for Kinetis & LPC MCU application development



Product Features

Architecture:
- CMSIS-CORE compatible
- Single driver for each peripheral
- Transactional APIs w/ optional DMA support for communication peripherals

Integrated RTOS:
- FreeRTOS v9
- RTOS-native driver wrappers

Integrated Stacks and Middleware:
- USB Host, Device and OTG
- Amazon Web Service IoT
- QCA WiFi Stacks
- USB Type-C Power Delivery Stack
- lwIP, FatFS
- Crypto acceleration plus wolfSSL & mbedTLS
- SD and eMMC card support

Reference Software:
- Peripheral driver usage examples
- Application demos
- FreeRTOS usage demos

License:
- BSD 3-clause for startup, drivers, USB stack

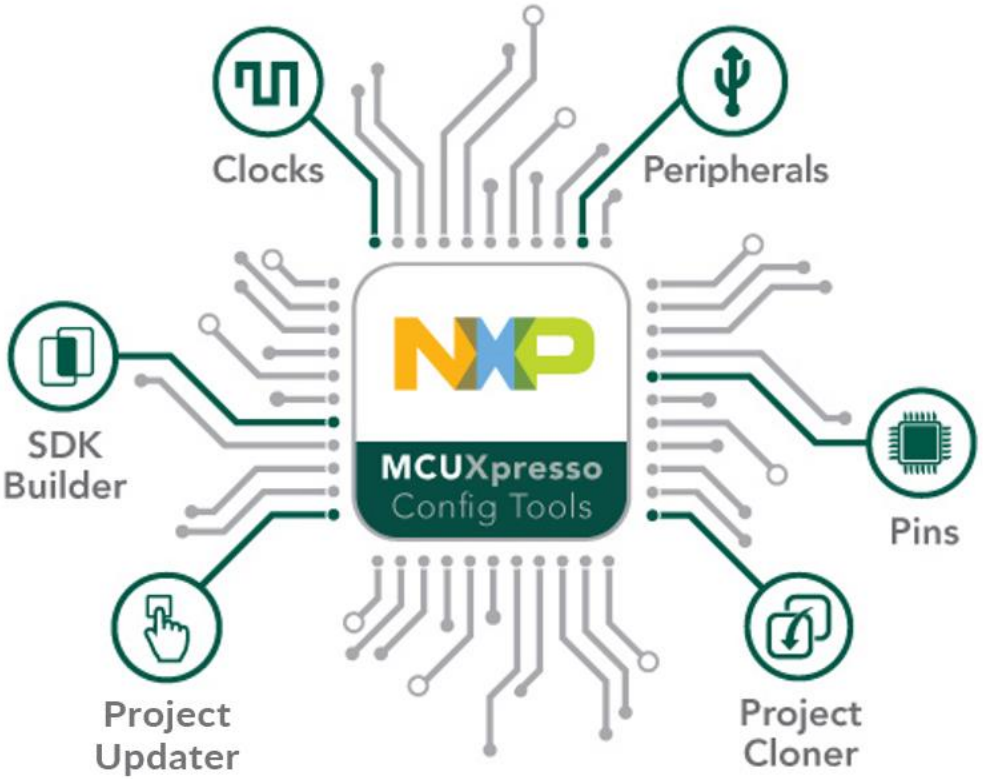
Project file support included:
- MCUXpresso IDE
- IAR®, ARM® Keil®, GCC w/ Cmake

Quality
- Production-grade software
- MISRA 2004 compliance
- Checked with Coverity® static analysis tools

MCUXpresso Config Tools

Learn more at: www.nxp.com/mcuxpresso/config

Integrated configuration and development tools for LPC and Kinetis MCUs



MCUXpresso Config Tools is a suite of evaluation and configuration tools that helps guide users from first evaluation to production software development.

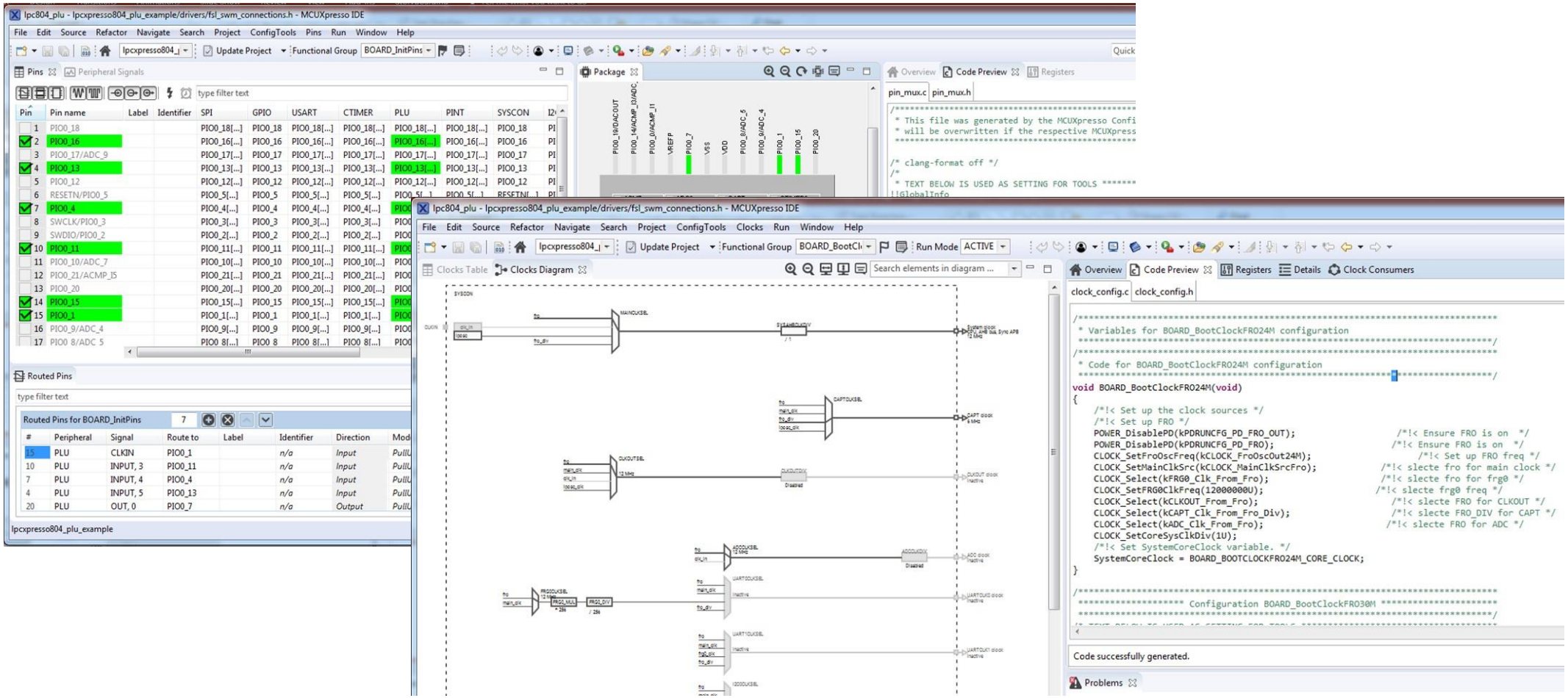
SDK Builder packages custom SDKs based on user selections of MCU, evaluation board, and optional software components.

Pins and **Clocks** tools generate initialization C code for custom board support. Features validation of inputs and cross-tool conflict resolution.

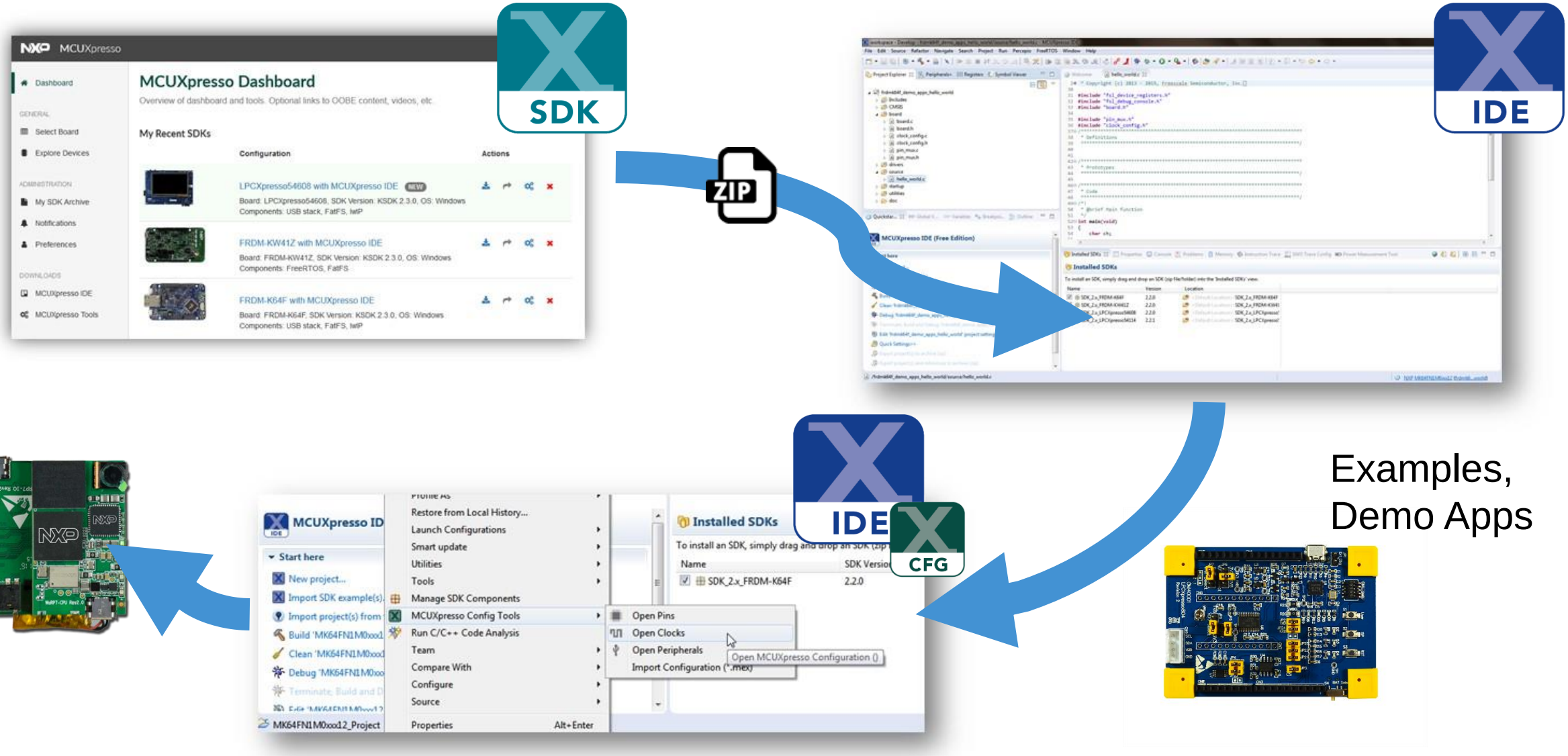
Project Update works directly with existing SDK-based IDE projects with generated Pins and Clocks source files

Project Cloning creates a standalone SDK project based on a example application available within SDK release.

Pin and clock config tools on LPC804 (example screenshot)



MCUXpresso Development flow



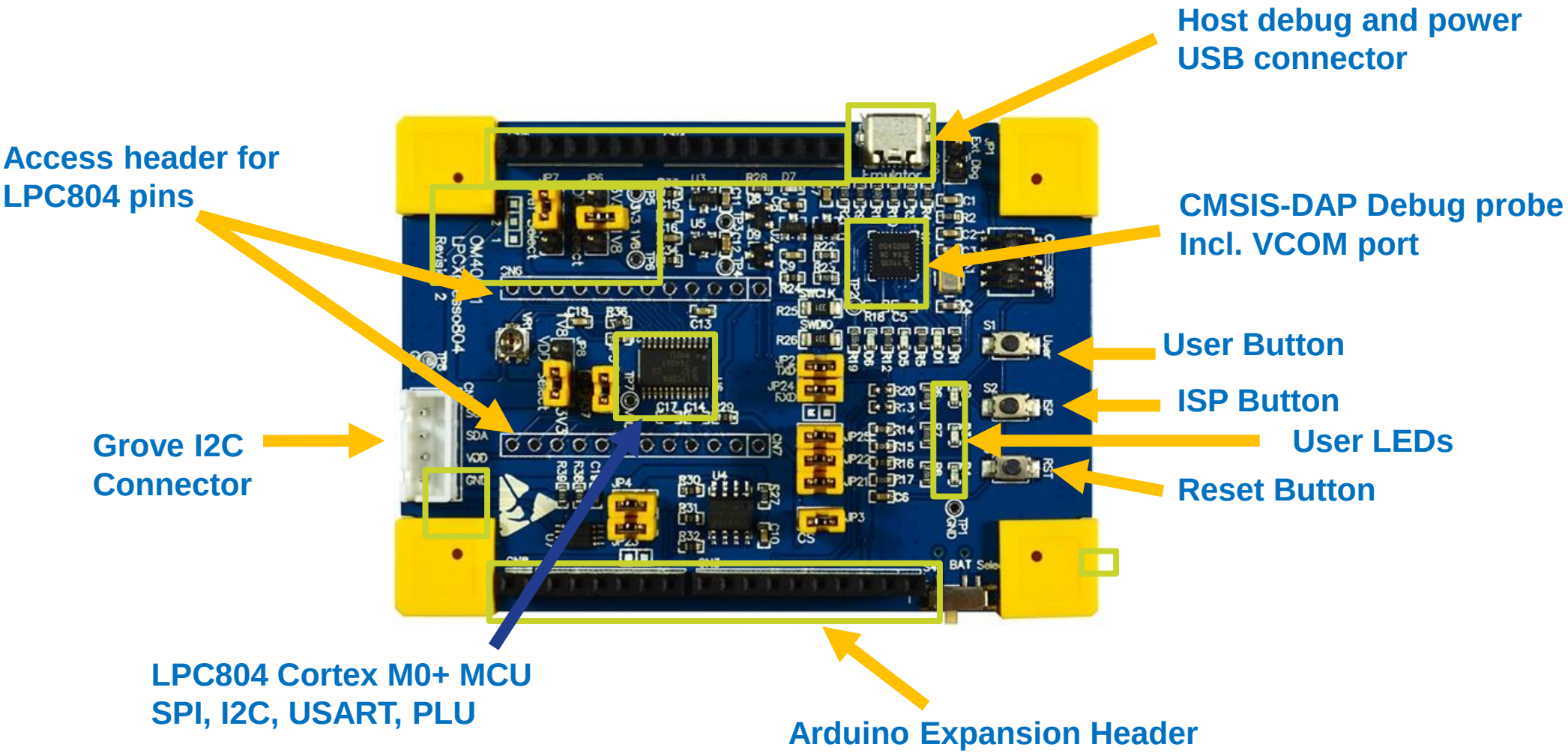
Code Bundles

- Available for all LPC8xx devices
- Project files included for MCUXpresso IDE, Keil and IAR
- Drivers and examples
- Simple, register-level examples
- Ideal for customers transitioning from 8 or 16 bit MCUs
- https://www.nxp.com/LPC800-Code-Bundles

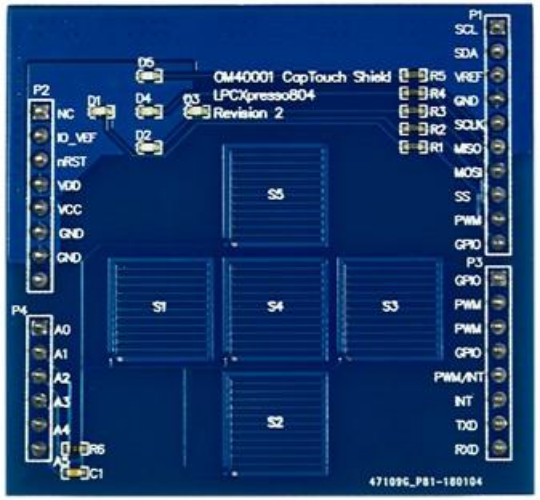
LPCXpresso804 Development Board

(OM40001)

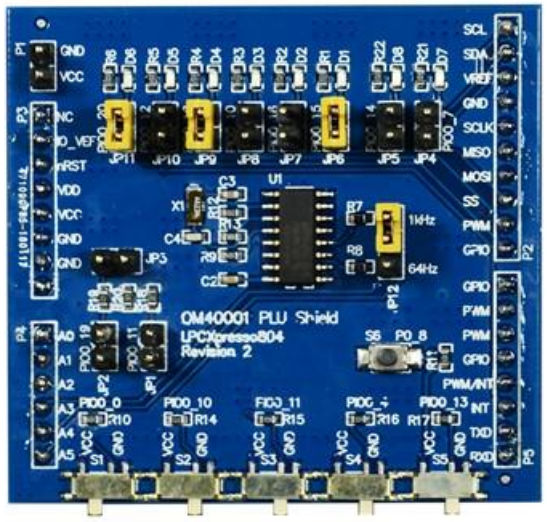
Learn more at: http://www.nxp.com/demoboard/om40001



LPCXpresso804 Dev Included Add-on Boards



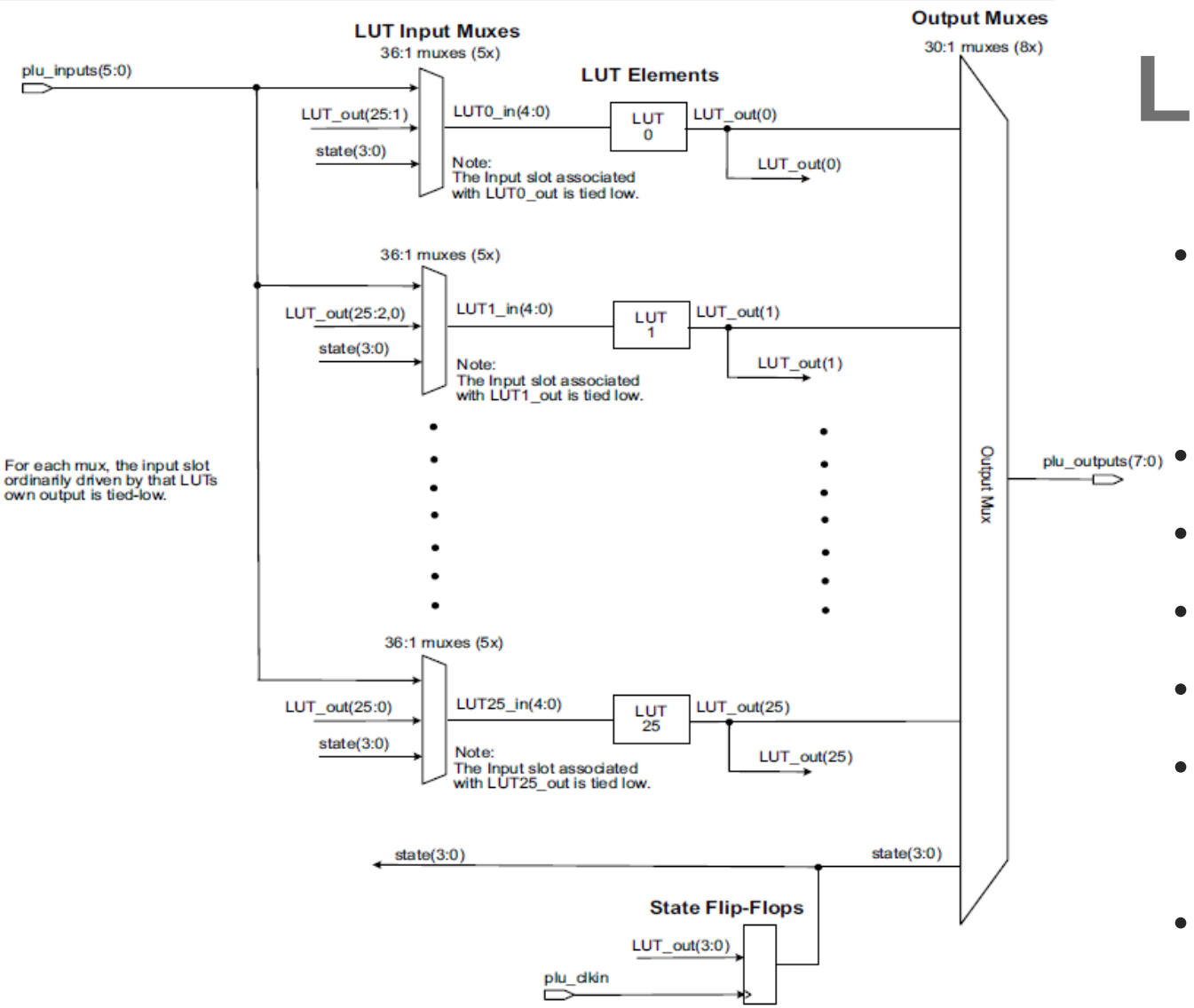
- Capacitive touch shield
  - 5 button array
  - LED associated with each button



- PLU shield
  - Selectable switch (high/low) for 5 inputs
  - Push button for 1 input
  - 1kHz / 8Hz oscillator
  - Selectable LEDs for all outputs

PROGRAMMABLE LOGIC UNIT (LPC804)

LPC804 PLU Features

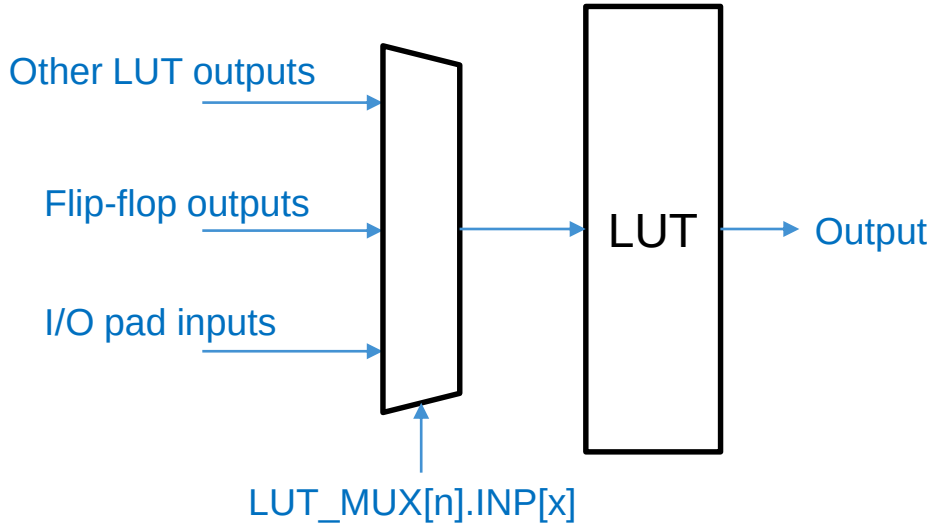


- Completely independent from MCU, which only configures logic via by registers
- Up to 6 PLU inputs from I/O pads
- Up to 8 outputs to I/O pads
- Up to 26 Look up tables (LUTs)
- Up to 4 state flip flops
- Clock input from external I/O pad

- Feedback to MCU via I/O pads or through switch matrix

Understanding Look Up Tables (LUTs)

- 26 LUTs in the PLU, which can implement any combinational function for up to 5 of the possible inputs
- Inputs:
  - other LUT outputs
  - state FF outputs
  - all inputs available from PLU I/O pads



Flip-flops in the PLU

- Four of the LUTs have an associated flip-flop (FF)
- Clock source is the same for all FFs, which is sourced from an external pin
- FFs can be used for counters, state machines, synchronization, etc.

Uses of the PLU

- Some potential functionality to implement in the PLU:
  - Integration of discrete logic device(s) to save cost and area
  - Event detection
  - Simple state machines
  - Stepper motor control
  - Generation of specialized waveforms / protocols in conjunction with other peripherals
- Benefits (vs function implementation on conventional MCU)
  - Low latency
  - Low power (can operate while MCU is in deep sleep)
  - Save MCU processing power for other tasks

PLU DESIGN TOOL

PLU Design Tool

- Free to download from nxp.com website
- Runs on Windows 7/8/10
- Generates C source code (register settings) to cut and paste into applications
- Three design methods available within the tool:
  - Direct LUT mapping
  - Schematic capture
  - Verilog

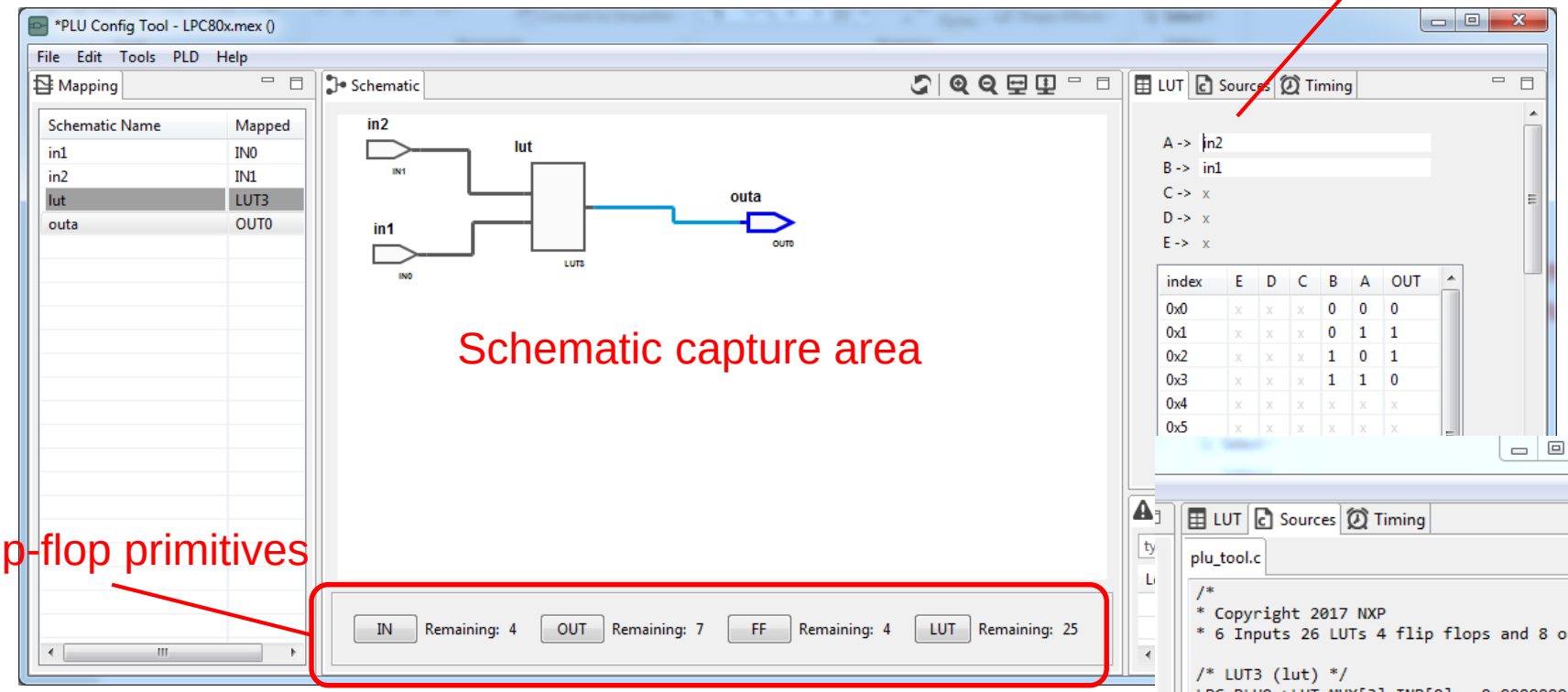
Design Methods – Direct LUT Mapping

- This method involves directly controlling the LUT configurations
- LUTs can be configured as standard logic gates, using a logic equation or using a truth table
- Register settings and timings are updated immediately, once all connections and LUTs/inputs/outputs are defined
- Benefits:
  - Simplest method, with clear view of how resources are used
- Disadvantages
  - Less efficient in use of resources than Verilog or schematic capture (relies on designer to manually optimize the design to minimize use of LUTs)
  - Difficult to implement functions like state machines (vs. Verilog)

Direct LUT Mapping

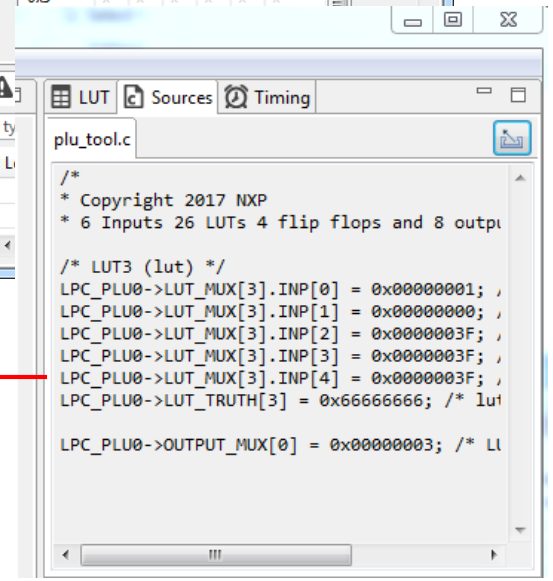
LUT configuration

Mapping of inputs, outputs and flops



Schematic capture area

LUT, I/O and Flip-flop primitives



Generated source

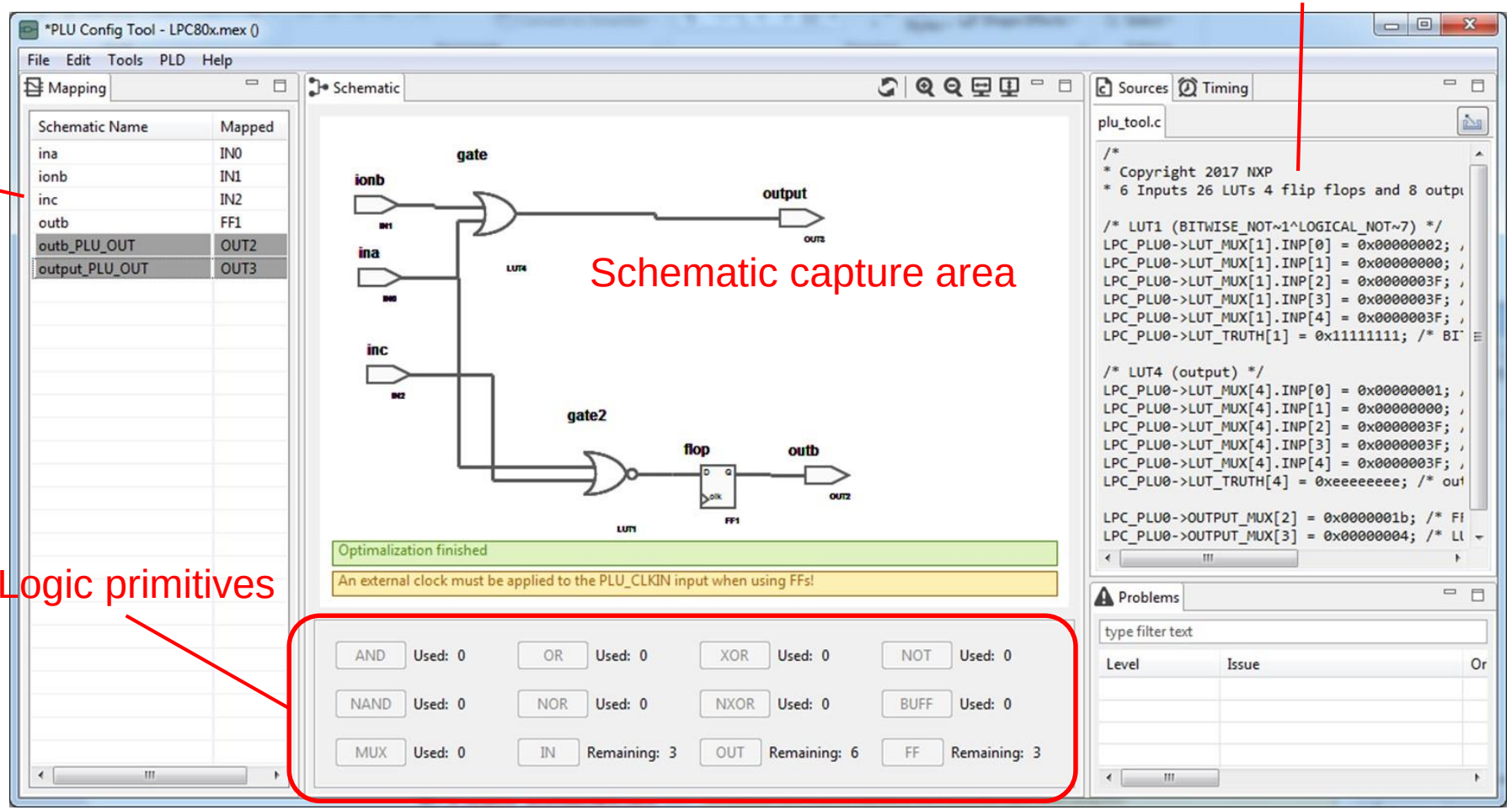
Design Methods – Schematic Capture

- Designs are implemented directly using standard gates and muxes
- Simple schematic capture tool
- Once design is ready, it can be synthesized/mapped to LUTs and other PLU resources
- Benefits
  - Easy migration of existing designs
  - Provides good optimization of design
- Disadvantages
  - More difficult to see exactly how a design has been mapped to the LUT hardware (register level settings)
  - More difficult to implement functions like state machines (vs. Verilog)

Schematic Capture

Register settings
(C format)

Mapping of inputs,
outputs and flops



Design Methods – Verilog

- Designs are synthesized/mapped to LUTs and other PLU resources from an input Verilog design
- Benefits
  - Easy migration of existing designs
  - Provides good optimization of design
  - Good for higher level functions such as state machines
- Disadvantages
  - Requires knowledge of Verilog HDL design
  - Designer needs to have an understanding of resources his/her design will need

SDK API functions for PLU setup

- SDK includes function calls for individual register setup, for more “readable” code
- Place in single set up function, so PLU tool output can be cut and pasted instead

```
static void PLU_SetLutAllInputOutputTruthTable(void)
{
        /* LUT0 (L1) */
        LPC_PLU0->LUT_MUX[0].INP[0] = 0x00000000; /* IN0 (in0) */
        PLU0->LUT_MUX[0].INP[1] = 0x0000003F; /* default */
        PLU0->LUT_MUX[0].INP[2] = 0x0000003F; /* default */
        PLU0->LUT_MUX[0].INP[3] = 0x0000003F; /* default */
        PLU0->LUT_MUX[0].INP[4] = 0x0000003F; /* default */
        PLU0->LUT_TRUTH[0] = 0x55555555; /* l0 (l0) STD 1 INPUT NOT */

        /* LUT1 (L2) */
        PLU0->LUT_MUX[1].INP[0] = 0x00000006; /* LUT0 (l1) */
        PLU0->LUT_MUX[1].INP[1] = 0x00000004; /* IN4 (i4) */
        PLU0->LUT_MUX[1].INP[2] = 0x0000003F; /* default */
        PLU0->LUT_MUX[1].INP[3] = 0x0000003F; /* default */
        PLU0->LUT_MUX[1].INP[4] = 0x0000003F; /* default */
        PLU0->LUT_TRUTH[1] = 0x88888888; /* L1 (L1) STD 2 INPUT AND */

        /* LUT2 (l2a) */
        PLU0->LUT_MUX[2].INP[0] = 0x00000005; /* IN5 (i5) */
        PLU0->LUT_MUX[2].INP[1] = 0x0000003F; /* default */
        PLU0->LUT_MUX[2].INP[2] = 0x0000003F; /* default */
        PLU0->LUT_MUX[2].INP[3] = 0x0000003F; /* default */
        PLU0->LUT_MUX[2].INP[4] = 0x0000003F; /* default */
        PLU0->LUT_TRUTH[2] = 0x55555555; /* l2 (l2) STD 1 INPUT NOT */

        PLU0->OUTPUT_MUX[0] = 0x00000000; /* LUT0 (l1) -> not_0_out */
        PLU0->OUTPUT_MUX[1] = 0x00000001; /* LUT1 (L2) -> i0_and_i4_out */
        PLU0->OUTPUT_MUX[2] = 0x0000001c; /* FF2 (ff) -> ff_out */

}
```

From PLU tool

```
static void PLU_SetLutAllInputOutputTruthTable(void)
{
    /* Select plu_input(3) to be connected to LUT0 input 0 */
    PLU_SetLutInputSource(PLU, kPLU_LUT_0, kPLU_LUT_IN_0, kPLU_LUT_IN_SRC_PLU_IN_3);
    PLU_SetLutTruthTable(PLU, kPLU_LUT_0, 0x55555555);
    /* Select LUT_out(0) to be connected to PLU output 0 */
    PLU_SetOutputSource(PLU, kPLU_OUTPUT_0, kPLU_OUT_SRC_LUT_0);

    /* Select LUT_out(0) to be connected to LUT1 input 0 */
    PLU_SetLutInputSource(PLU, kPLU_LUT_1, kPLU_LUT_IN_0, kPLU_LUT_IN_SRC_LUT_OUT_0);
    /* Select plu_input(4) to be connected to LUT1 input 1 */
    PLU_SetLutInputSource(PLU, kPLU_LUT_1, kPLU_LUT_IN_1, kPLU_LUT_IN_SRC_PLU_IN_4);
    PLU_SetLutTruthTable(PLU, kPLU_LUT_1, 0x88888888);
    /* Select LUT_out(1) to be connected to PLU output 1 */
    PLU_SetOutputSource(PLU, kPLU_OUTPUT_1, kPLU_OUT_SRC_LUT_1);

    /* Select plu_input(5) to be connected to LUT2 input 0 */
    PLU_SetLutInputSource(PLU, kPLU_LUT_2, kPLU_LUT_IN_0, kPLU_LUT_IN_SRC_PLU_IN_5);
    PLU_SetLutTruthTable(PLU, kPLU_LUT_2, 0xaaaaaaaa);
    /*
     * Select Flip-Flops state(2) to be connected to PLU output 2
     * Note: An external clock must be applied to the PLU_CLKIN input when using FFs.
     */
    PLU_SetOutputSource(PLU, kPLU_OUTPUT_2, kPLU_OUT_SRC_FLIPFLOP_2);
}
```

Using SDK API

ODIN II and RASP: Synthesis Engine in the PLU Tool

- ODIN II
  - Odin II is used for logic synthesis and elaboration, converting a subset of the Verilog Hardware Description Language (HDL) into a BLIF netlist
  - Open source
  - Windows version installed by default by PLU Tool installer
- RASP
  - Mapping and synthesis package developed by VLSI lab at UCLA
  - Provided under license agreement for free commercial use on NXP devices
  - Windows version, ported by NXP, is installed by default by PLU Tool installer

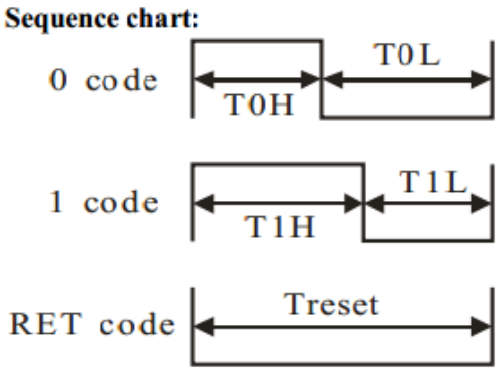
EXAMPLE APPLICATIONS

Example Applications

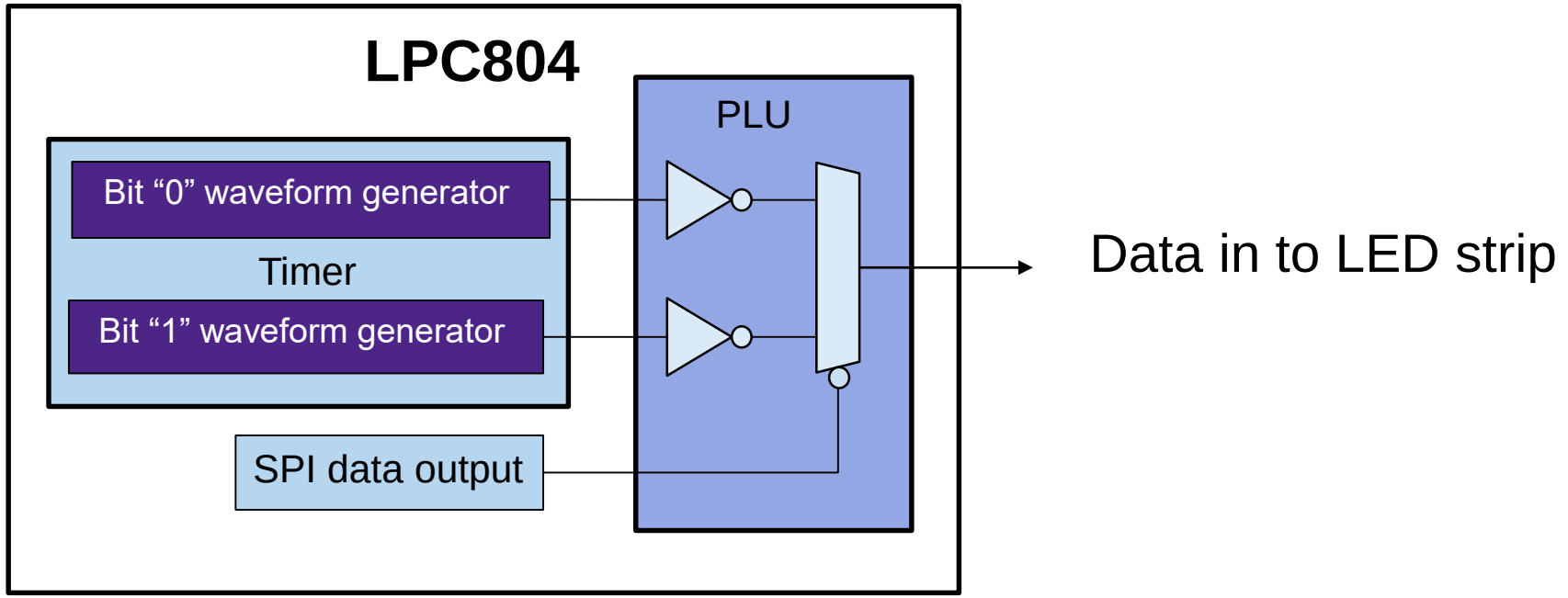
- WS2812 LED control
- Stepper motor control
- DC motor driver
- Pattern generator

WS2812 RGB LED Driver

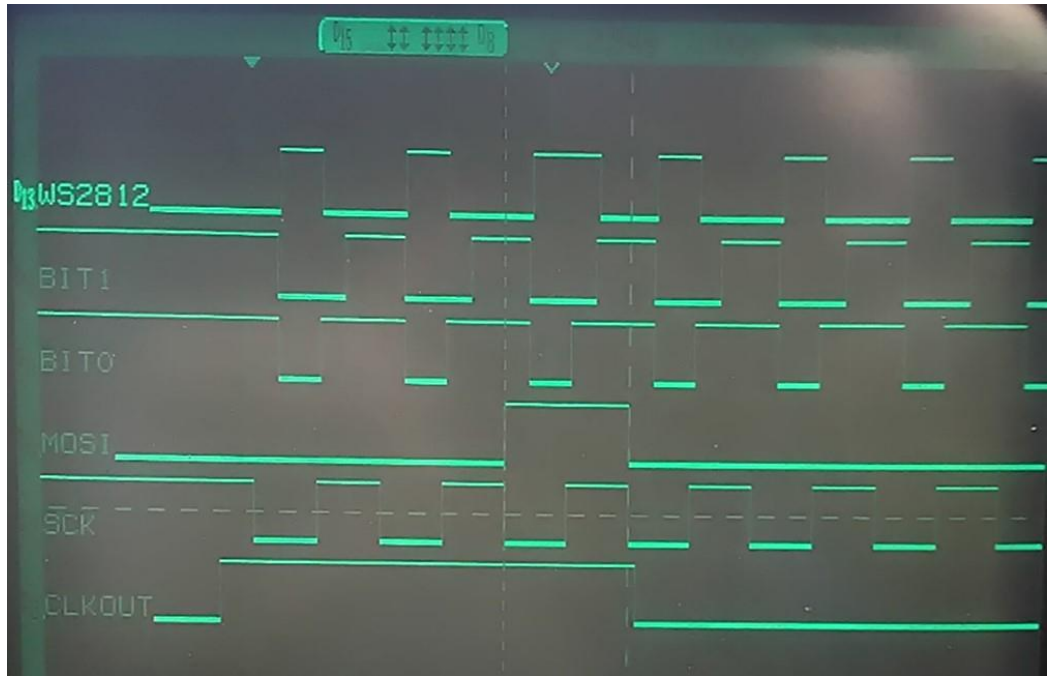
- Daisy-chainable RGB LEDs, requiring only +5V, GND and serial data
- Simple protocol, but not directly supported by an “standard” MCU protocol
- PLU design utilizes SPI data output to select between “0” or “1” data bit waveform
- Very low overhead MCU management required, no complex timing to handle

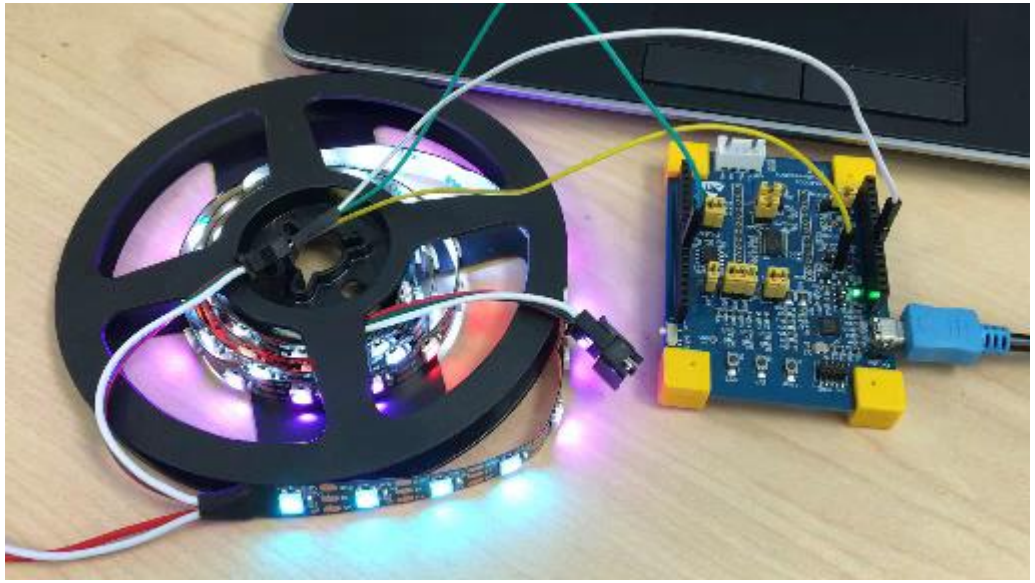


WS2812 RGB LED Driver – Simplified schematic



WS2812 Driver – Waveforms / running on LPCXpresso804





WS2812 Driver Code (excerpts)

SPI ISR

```
//continue sending data (if any left) or disable interrupts and end
the transfer
void SPI0_IRQHandler(void)
{
      SPI0->TXDAT = *(pnt_8bit_data+ws2812_tx_index);//send next
                                                      data
      ws2812_tx_index++;                            //update index
      if (ws2812_tx_index == ws2812_tx_cnt)
      {
            SPI0->INTENCLR = SPI_INTENCLR_TXRDYEN(1); //in case
                                                      no more data left
                                                      disable TXD int

      }
      return;
}
```

Ptr to LED value global array

Total # LEDs

LED update code in main() loop

```
…
while(1)
   {
//sequentially update WS2812 LEDs to red
   for (i = 0; i != WS2812_LED_CNT; i++)
   {
            ws2812_data[i].red   = WS2812_OUTLVL;
            ws2812_data[i].green = 0x00;
            ws2812_data[i].blue  = 0x00;
            start_transfer(1);
            delay_systick(SYSTICK_12MHZ_DELAY_200MS);
    }
…
```

Global array of LEDvalues

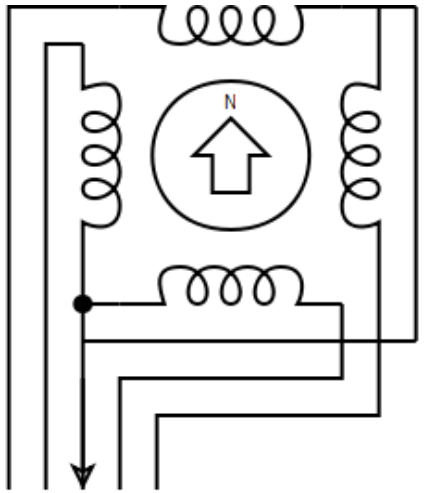
LED value update function

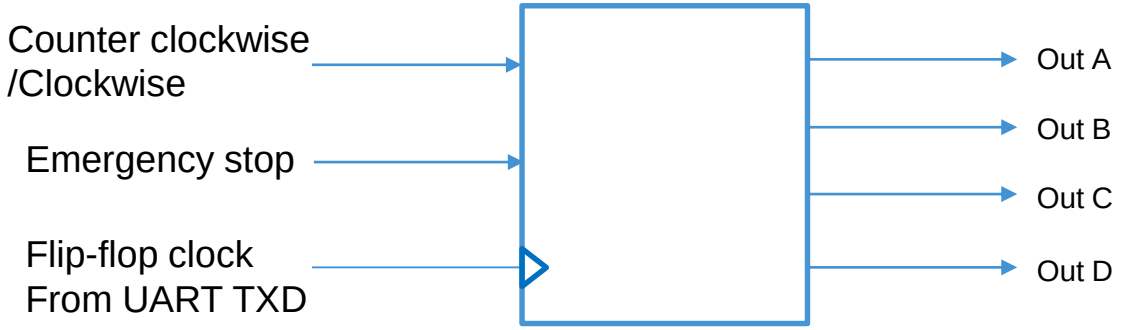
```
// input parameter = 0: do not wait for the data to be latched by WS2812(s)
//                   1: wait for the data to be latched by WS2812(s)
void start_transfer(uint32_t wait_for_done)
{
      while((CTIMER0->TCR & CTIMER_TCR_CEN_MASK) != 0); //ensure last transfer ended
      ws2812_tx_index = 0;                            //clear the tx data index
      CTIMER0->TC = CTIMER0->MR[2]+1;                 //update the TC
      CTIMER0->TCR = CTIMER_TCR_CEN(1);               //let the timer run
      SPI0->INTENSET = SPI_INTENSET_TXRDYEN(1);       //enable TXDAT based interrupt

      if (wait_for_done != 0)                         //wait for the timer to stop...
      {//...
            while((CTIMER0->TCR & CTIMER_TCR_CEN_MASK) != 0);//...
      }//...

       return;
}
```

- Initialization excluded, but simply sets up PLU, CTIMER, SPI interface and clock
- start_transfer function called for each update of the LED array

Unipolar Stepper Motor Control

- 8 half step control of a unipolar stepper
- Simple direction control input
- Utilizes UART to generate step pulses, easy speed control and fast operation
- Emergency stop input to PLU enables motor to be stopped in 1 step time
- Very low overhead to MCU (services UART interrupts every 4 steps)





Stepper Motor Control Implementation

Basic state machine
– 8 half step states moving to next step

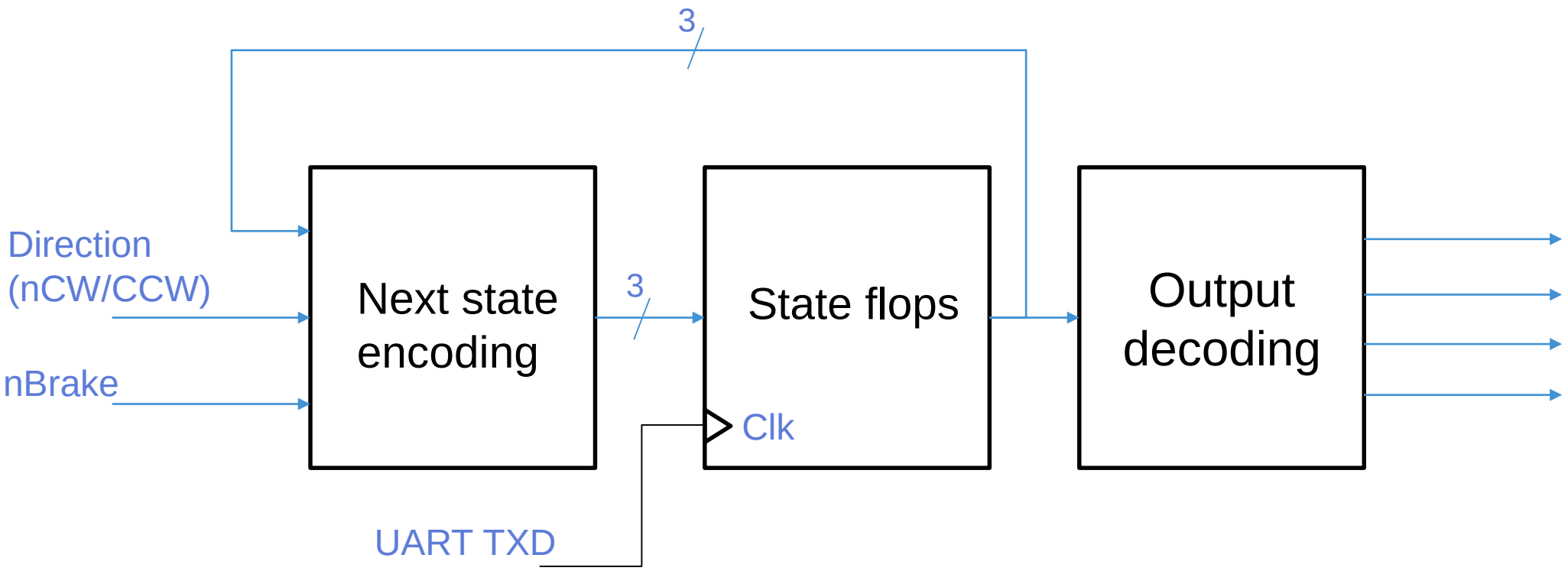
CCW next state (DOWN)				current state				CW next state (UP)		
bit2	bit1	bit0		FF2	FF1	FF0		bit2	bit1	bit0
1	1	1		0	0	0		0	0	1
0	0	0		0	0	1		0	1	0
0	0	1		0	1	0		0	1	1
0	1	0		0	1	1		1	0	0
0	**1**	**1**	<=	**1**	**0**	**0**	=>	**1**	**0**	**1**
1	0	0		1	0	1		1	1	0
1	0	1		1	1	0		1	1	1
1	1	0		1	1	1		0	0	0

	nCW/CCW	next state			
FF	bit3	bit2	bit1	bit0	
000	0	0	0	1	CW
001		0	1	0	
010		0	1	1	
011		1	0	0	
100		1	0	1	
101		1	1	0	
110		1	1	1	
111		0	0	0	
000	1	1	1	1	CCW
001		0	0	0	
010		0	0	1	
011		0	1	0	
100		0	1	1	
101		1	0	0	
110		1	0	1	
111		1	1	0	

Adding direction control and emergency brake

	nBrake	nCW/CCW	next state			
FF	bit4	bit3	bit2	bit1	bit0	
000	0	0	0	0	0	bit = FF
001	0	0	0	0	1	
010	0	0	0	1	0	
011	0	0	0	1	1	
100	0	0	1	0	0	
101	0	0	1	0	1	
110	0	0	1	1	0	
111	0	0	1	1	1	
000	0	1	0	0	0	
001	0	1	0	0	1	
010	0	1	0	1	0	
011	0	1	0	1	1	
100	0	1	1	0	0	
101	0	1	1	0	1	
110	0	1	1	1	0	
111	0	1	1	1	1	
000	1	0	0	0	1	CW
001	1		0	1	0	
010	1		0	1	1	
011	1		1	0	0	
100	1		1	0	1	
101	1		1	1	0	
110	1		1	1	1	
111	1		0	0	0	
000	1	1	1	1	1	CCW
001	1		0	0	0	
010	1		0	0	1	
011	1		0	1	0	
100	1		0	1	1	
101	1		1	0	0	
110	1		1	0	1	
111	1		1	1	0	

Stepper Motor Controller – Design Structure



Code Excerpts – Stepper Motor Controller

```
void USART1_IRQHandler(void)
{
      GPIO->SET[usart1_isr_port] = 1<<usart1_isr_pin;

      smc_clks_isr++;
      if (smc_clks_left >= 5)
      {
            USART1->TXDAT = 0x55;
            smc_clks_left -= 5;
      }
      else
      {
      switch(smc_clks_left)
      {
         default:
          case 0: USART1->INTENCLR = USART_INTENCLR_TXRDYCLR(1);
                  NVIC_ClearPendingIRQ(USART1_IRQn);
              break;
          case 1: USART1->TXDAT = 0xFF;
                  break;
          case 2: USART1->TXDAT = 0xFD;
                  break;
          case 3: USART1->TXDAT = 0xF5;
                  break;
          case 4: USART1->TXDAT = 0xD5;
                  break;
      }
      smc_clks_left = 0;
      }
      GPIO->CLR[usart1_isr_port] = 1<<usart1_isr_pin;

return;
}
```

Transmit 0x55 as clock pattern to PLU flops

As we get to <5 clks change output to ensure we only get desired number of clk edges

UART ISR

```
void setup_smc_clkgen(uint32_t brg, uint32_t hs_count)
{
      //CFG: 1 STOP, no parity, 8 data; divide by 12 (1 MHz out of 12 MHz),
      stepper clk =...
      USART1->CFG = USART_CFG_STOPLEN(0) | USART_CFG_PARITYSEL(0) |
      USART_CFG_DATALEN(1) | USART_CFG_ENABLE(0);
      USART1->BRG = USART_BRG_BRGVAL(brg-1);
      USART1->CFG = USART_CFG_STOPLEN(0) | USART_CFG_PARITYSEL(0) |
      USART_CFG_DATALEN(1) | USART_CFG_ENABLE(1);

      smc_clks_left = hs_count;
      USART1->INTENSET = USART_INTENSET_TXRDYEN(1);

      return;
}
```

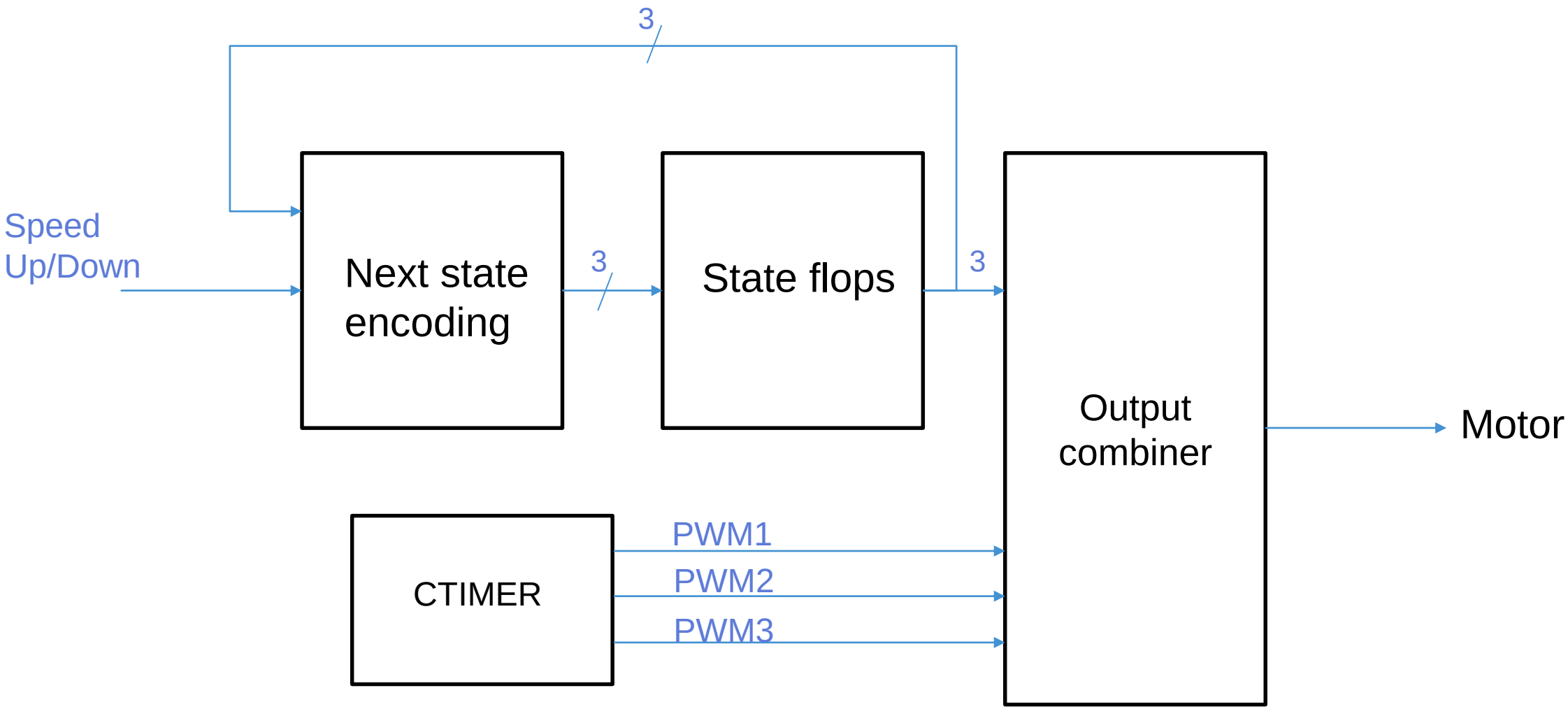
Simple routine to set up half step rate and count

```
setup_smc_clkgen(STEPPER_HSTEP_RATE_1000HZ, 4096);
```

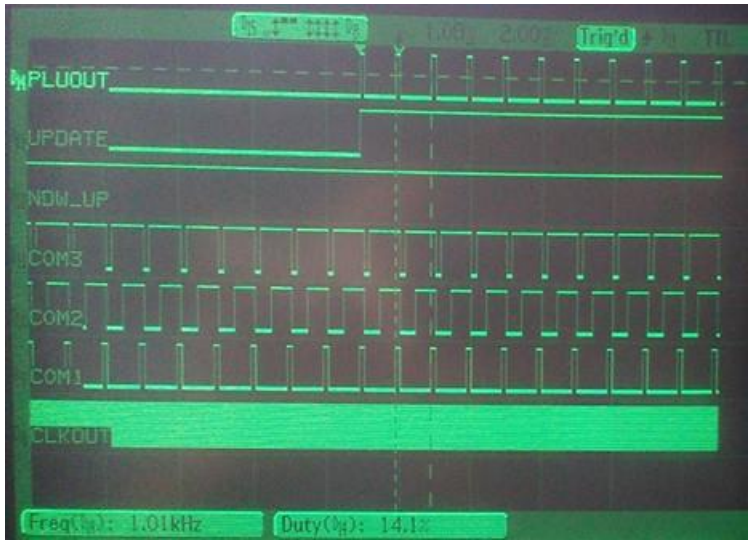
DC Motor Control

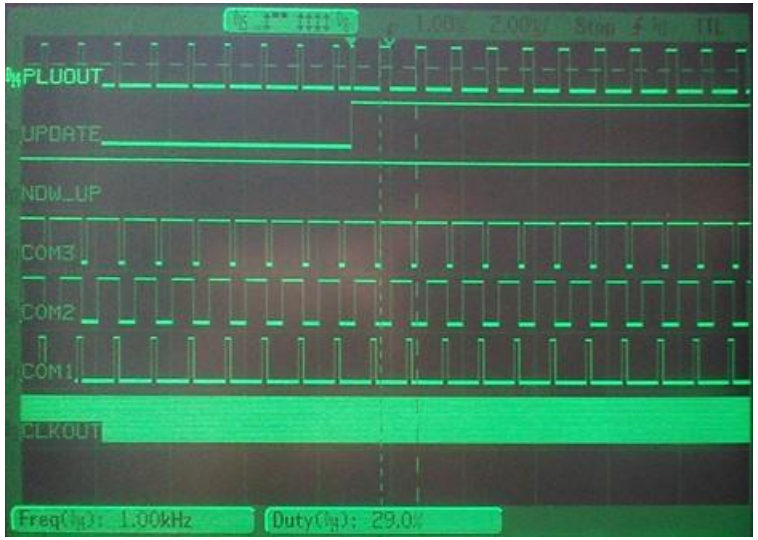
- This design augments to the CTIMER using the PLU for better PWM control
- 3 PWM rates combined to provide granular control from 14% to 100% duty cycle in 7 steps
- PLU also provides directional control signal
- Zero overhead for MCU after PLU & Timer configured
  - MCU control via GPIO could be added in application

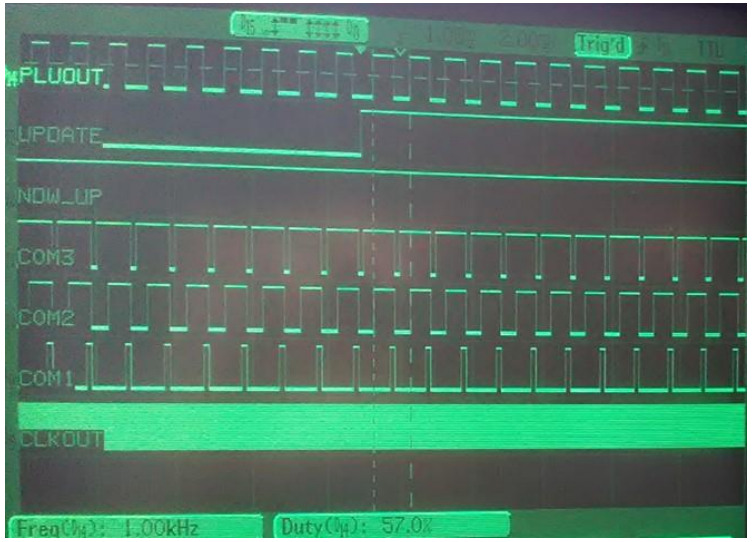
DC Motor Controller – Design Structure



DC Motor Control Waveforms (speed increasing)

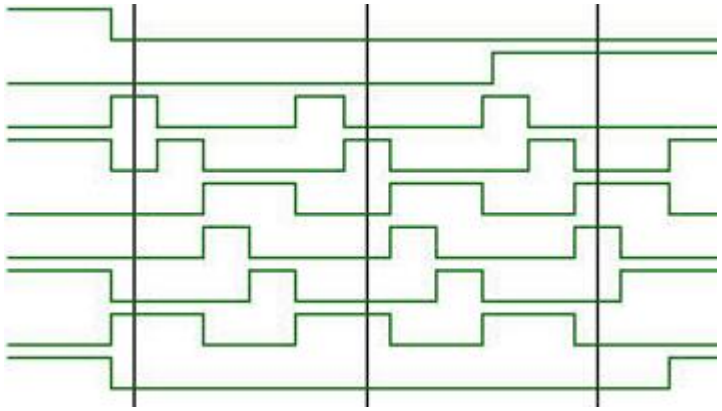




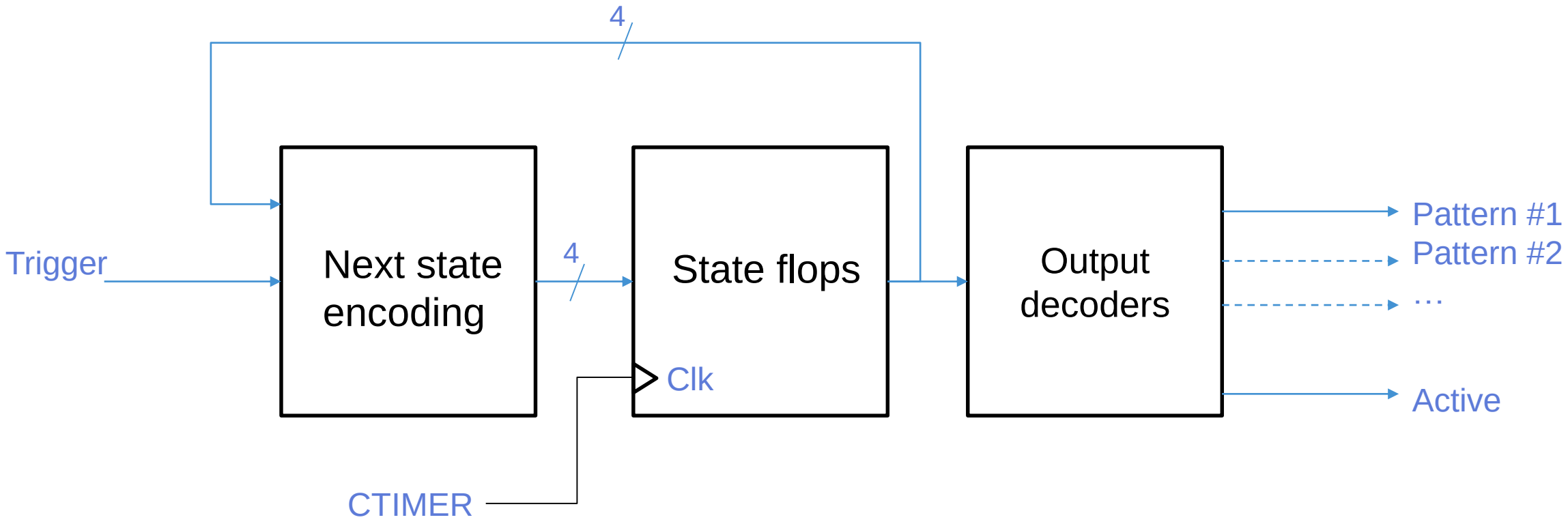


Pattern Generator

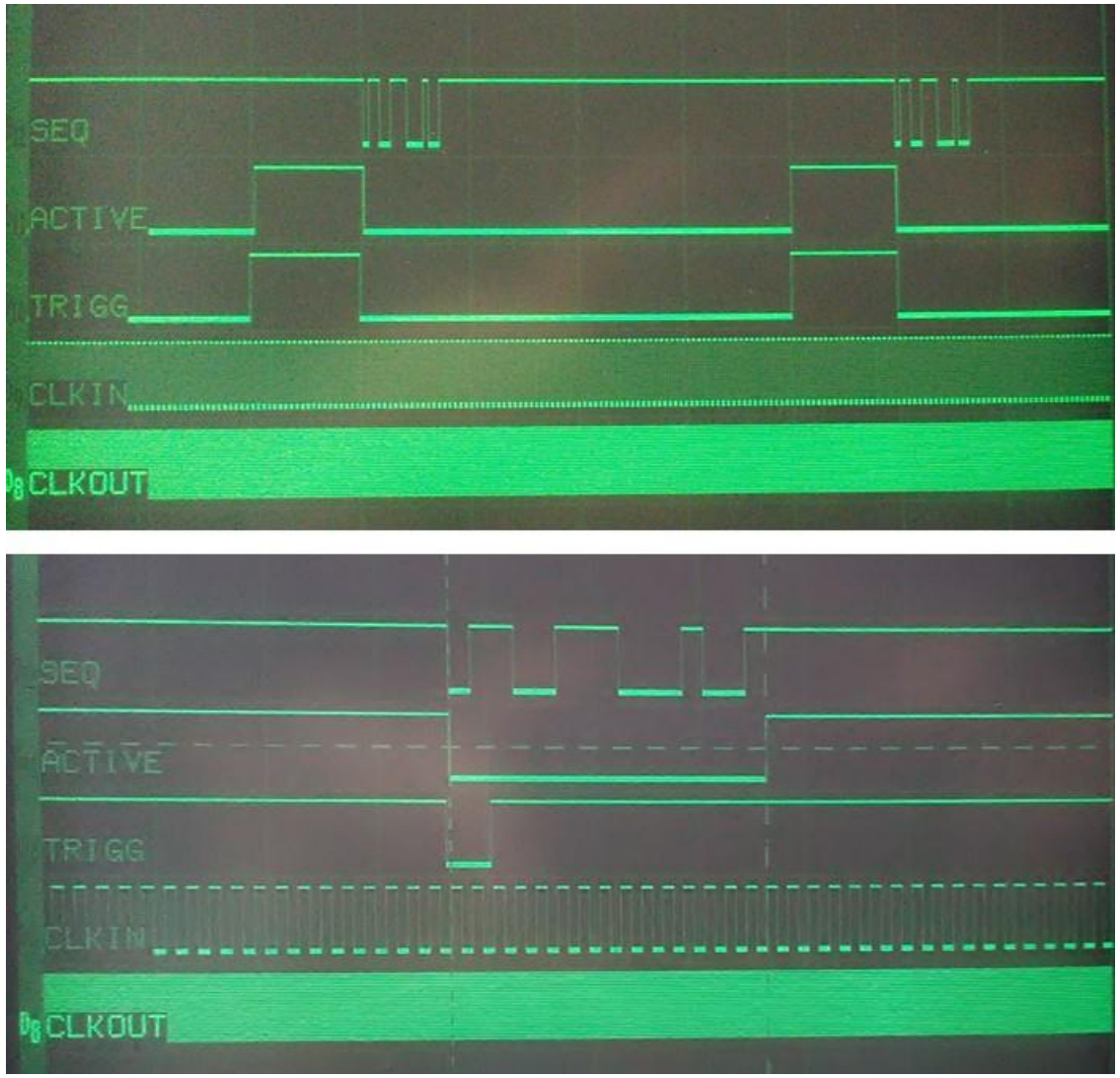
- 15-bit pattern generation
- Trigger assert/de-assert controls output
- Any pattern combination possible for available output pins
- State machine based
- Zero overhead for MCU after PLU and Timer setup
  - MCU control via GPIO can be added in applications, as needed



Pattern Generator – Design Structure



Pattern Generator – Design Structure



CONCLUSION

Recap and Conclusions

- LPC804 PLU has multiple uses in a wide range situations
- PLU block operates autonomously once set up via registers
- Very simple to use, yet powerful
  - Mop up glue logic
  - Extend capabilities of LPC804 peripherals
  - Generate custom waveforms
  - Very low to zero overhead for LPC804 MCU, with optimal response to external signals
- Free tools provide options for direct, schematic or Verilog based design methods
- LPC804 supported by popular 3rd party tools and free MCUXpresso tool suite

MCUXpresso Software and Tools Resources

Web pages

- **MCUXpresso Software and Tools – www.nxp.com/mcuxpresso**
- MCUXpresso SDK – www.nxp.com/mcuxpresso/sdk
- MCUXpresso IDE – www.nxp.com/mcuxpresso/ide
- MCUXpresso Config Tools – www.nxp.com/mcuxpresso/config

Supported Devices: Supported Devices Table (Community Doc)

Communities

- **MCUXpresso Software and Tools - https://community.nxp.com/community/mcuxpresso**
- MCUXpresso SDK- https://community.nxp.com/community/mcuxpresso/mcuxpresso-sdk
- MCUXpresso IDE- https://community.nxp.com/community/mcuxpresso/mcuxpresso-ide
- MCUXpresso Config Tools -https://community.nxp.com/community/mcuxpresso/mcuxpresso-config