TB3273

Getting Started with CLC on PIC18

Introduction

Author: Gheorghe Turcan, Microchip Technology Inc.

Using logic gates, systems can make decisions based on criteria without Central Processing Unit (CPU) intervention. Traditionally, these logic gates are manually implemented as external components. However, the PIC® microcontrollers (MCUs) offer this as a built-in functionality.

The Configurable Logic Cell (CLC) has a variety of basic gates, as well as sequential logic options that can be customized to create the logic specific to many applications. Using these logic gates, the CLC gives the ability of combining signals to make a new custom signal without running code to execute it. The CLC also offers flexible input selection (internal, external signals) and configurable output for internal or external use with the help of the Peripheral Pin Select (PPS).

This technical brief explains the concepts of the CLC and its implementation in the PIC18 family of microcontrollers with the following use cases:

- **Using Basic Logic Gates:**
  This example shows how to use the CLC to implement the basic logic functions AND, OR and XOR of two internally generated signals. The outputs of the CLCs are connected to I/O pins.
- **Using CLCs to Create a Data Signal Modulator:**
  This example shows how to use the CLC in both J-K flip-flop and AND-OR configurations to obtain a DSM function (multiple frequency carrier signal).
- **Using the CLC to Create an LED Dimming Effect:**
  This example shows how to use the CLC in an SR Latch configuration to obtain a PWM signal that changes duty cycle without code addition. The change in duty cycle can be used to generate the effect of an LED dimming or increasing brightness instead of blinking.

Note: For each use case, there are two different implementations that have the same functionalities: one code generated with MPLAB® Code Configurator (MCC) and one bare metal code. The MCC generated code offers hardware abstraction layers that ease the use of the code across different devices from the same family. The bare metal code is easier to follow and allows a fast ramp-up on the use case associated code.

View Code Examples on GitHub
Click to browse repositories

Table of Contents

1. Peripheral Overview

The Configurable Logic Cell (CLC) module provides programmable logic that operates outside the speed limitations of software execution. The CLC takes up to 64 input signals and, using configurable gates, reduces the 64 inputs to four logic lines that drive one of eight selectable single-output logic functions.

Input sources are a combination of the following components:

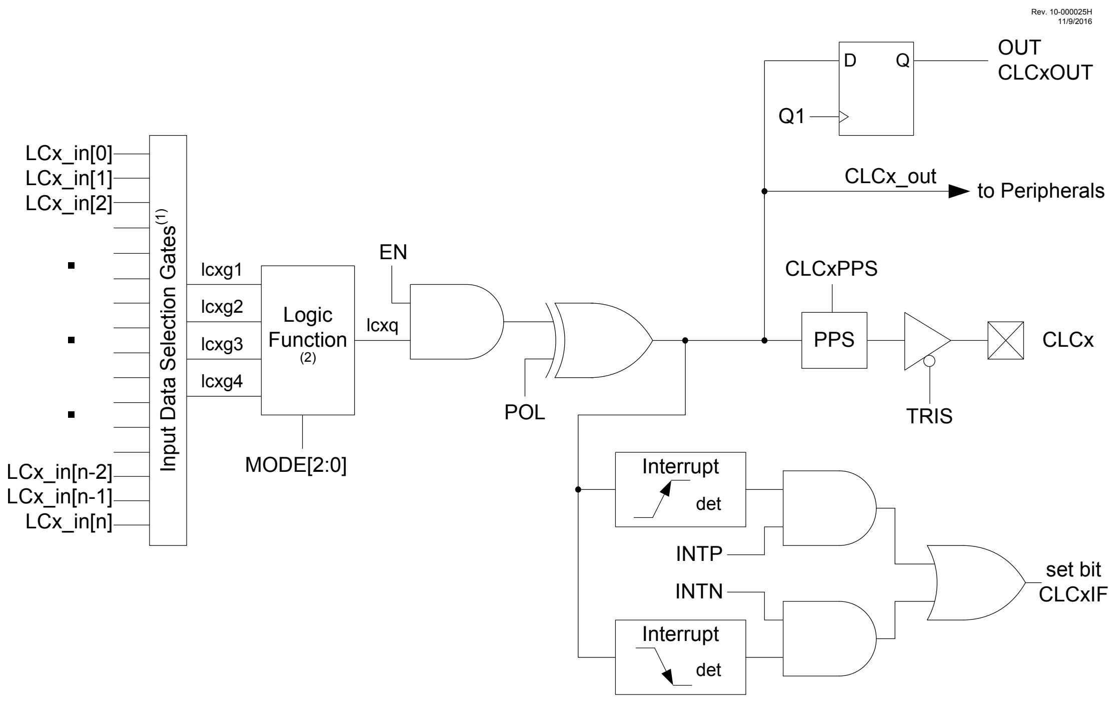
- I/O pins
- Internal clocks
- Peripherals
- Register bits

The output can be directed internally to peripherals and to an output pin.

Important: There are several CLC instances on this device. Throughout this section, the lower case x in register names is a generic reference to the CLC instance number (CLCx). For example, the first instance of the control register is CLC1CON and is generically described in this chapter as CLCxCON.

The following figure is a simplified diagram showing signal flow through the CLC.

Figure 1-1. CLC Simplified Block Diagram

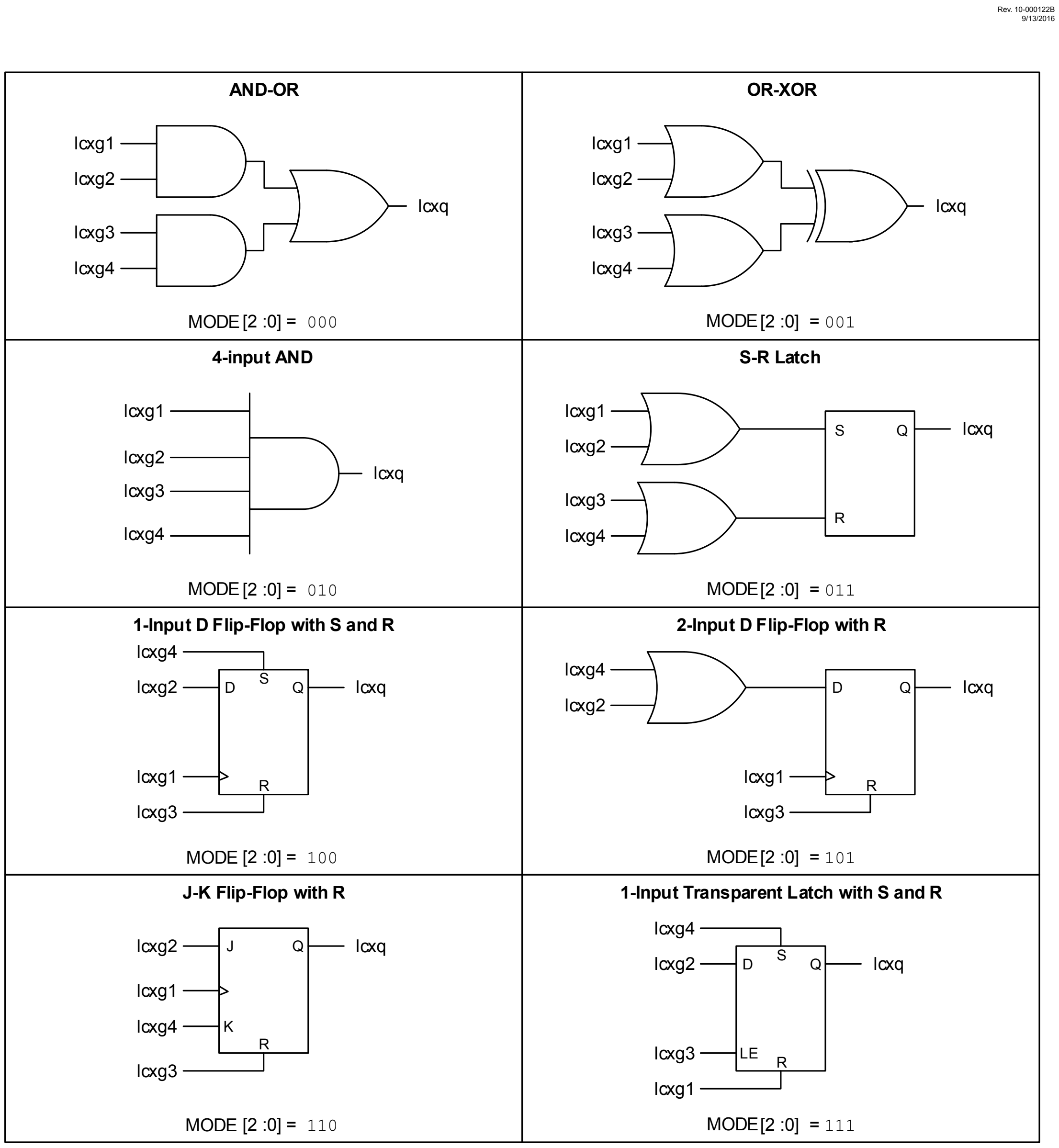


Possible configurations include:

- Combinatorial Logic:
  - AND
  - NAND
  - AND-OR
  - AND-OR-INVERT
  - OR-XOR

  - OR-XNOR
- Latches:
  - S-R
  - Clocked D with Set and Reset
  - Transparent D with Set and Reset
  - Clocked J-K with Reset

Figure 1-2. Programmable Logic Functions



Programming the CLC module is performed by configuring the four stages in the logic signal flow. These stages are:

- Data selection
- Data gating
- Logic function selection
- Output polarity

Each stage is setup at run time by writing to the corresponding CLC Special Function Registers. This has the added advantage of permitting logic reconfiguration on-the-fly during program execution.

2. Using Basic Logic Gates

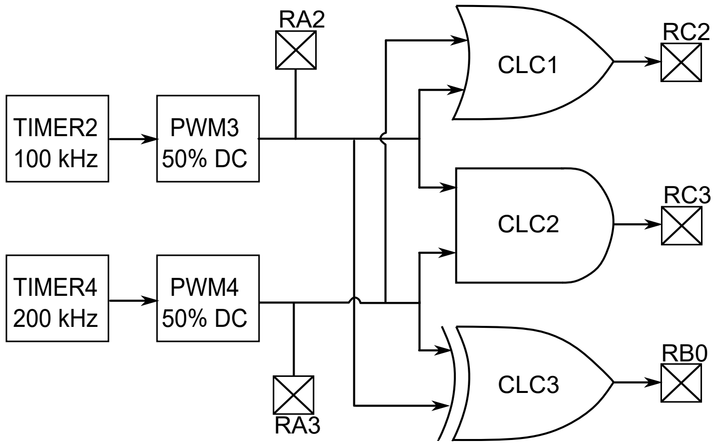
This example shows a basic initialization of three CLC peripherals in the Logic Gate mode and how to link to internal peripherals and to I/O ports. The three basic logic functions to take into consideration are: AND, OR and XOR.

The input signals going through the CLC are generated internally by the Timer2, Timer4, PWM3 and PWM4 peripherals. The outputs of the CLC and PWMs are connected to I/O ports for oscilloscope measurements as shown in Figure 2-1.

Setup configurations are the following:

- Timer2 frequency = 100 kHz (10 us period)
- Timer4 frequency = 200 kHz (5 us period)
- PWM3 has Timer2 as source and duty cycle = 50%
- PWM4 has Timer4 as source and duty cycle = 50%
- CLC1 is set up as AND-OR: used as 2-input OR
- CLC2 is set up as 4-input AND: used as 2-input AND
- CLC3 is set up as OR-XOR: used as 2-input XOR

Figure 2-1. Internal Connections for "Using Basic Logic Gates" Example



The following are the truth tables and oscilloscope results of each CLC:

1. For the OR example (CLC1), two input signals are used. The truth table is depicted below:

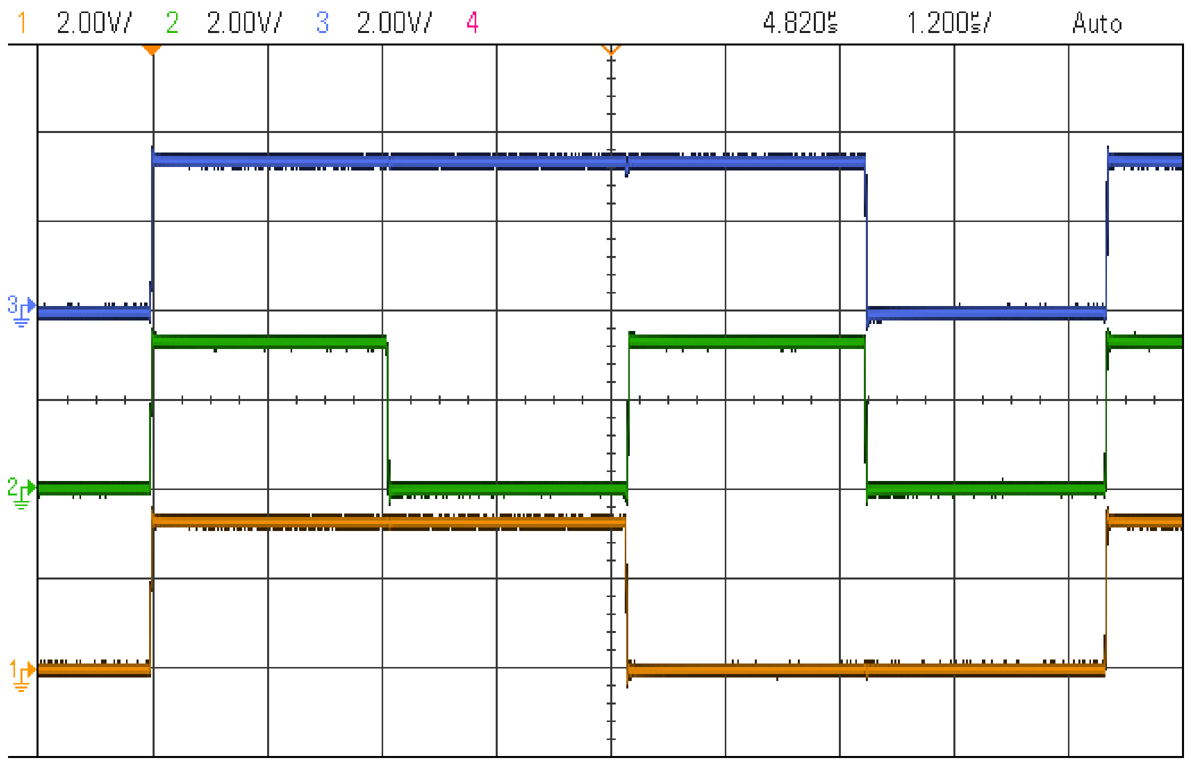
Table 2-1. Truth Table for 2-Input Logic OR Function

A	B	O = A OR B
0	0	0
0	1	1
1	0	1
1	1	1

Figure 2-2 depicts the input and output signals of CLC1 as it performs the OR logic function:

– Signal 1 (Orange) is PWM3 output used as CLC1 Input A
– Signal 2 (Green) is PWM4 output used as CLC1 Input B
– Signal 3 (Blue) is CLC1 output (logic OR between PWM3 and PWM4)

Figure 2-2. Oscillograms of CLC1 Performing Logic OR, Input Signals PWM3 and PWM4



2. For the AND example, two input signals are used (the CLC in AND mode can have up to four inputs). The truth table is shown below:

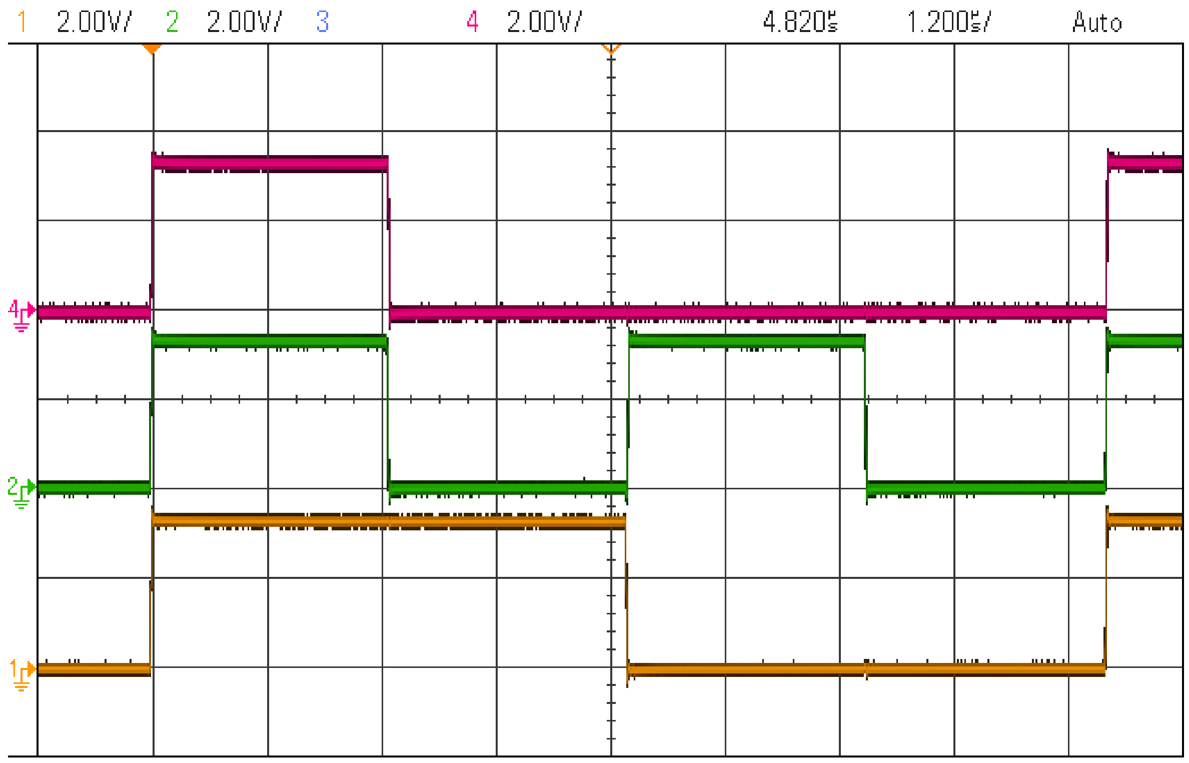
Table 2-2. Truth Table for 2-Input Logic AND Function

A	B	O = A AND B
0	0	0
0	1	0
1	0	0
1	1	1

Figure 2-3 depicts the input and output signals of CLC2 as it performs the AND logic function:

– Signal 1 (Orange) is PWM3 output used as CLC2 Input A
– Signal 2 (Green) is PWM4 output used as CLC2 Input B
– Signal 4 (Red) is CLC2 output (logic AND between PWM3 and PWM4)

Figure 2-3. Oscillograms of CLC2 Performing Logic AND, Input Signals PWM3 and PWM4



3. For the XOR example, two input signals are used. The truth table is shown below:

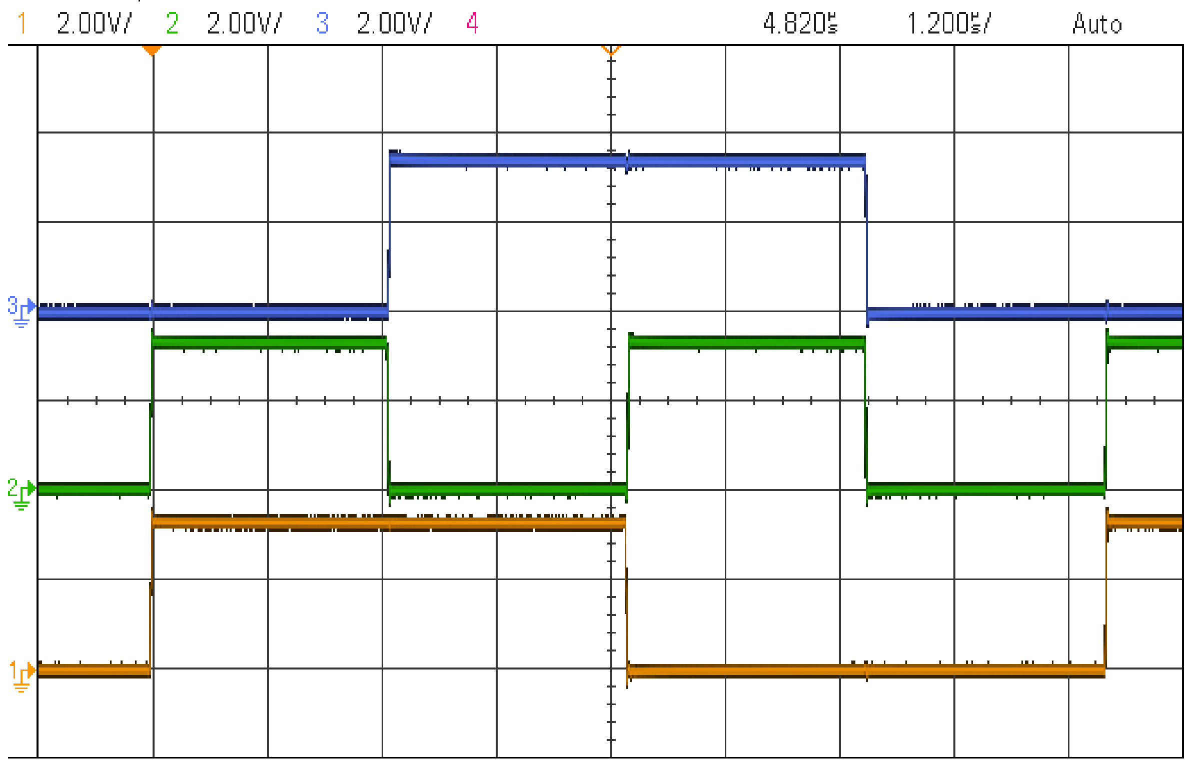
Table 2-3. Truth Table for 2-Input Logic XOR Function

A	B	O = A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Figure 2-4 depicts the input and output signals of CLC3 as it performs the XOR logic function:

– Signal 1 (Orange) is PWM3 output used as CLC3 Input A
– Signal 2 (Green) is PWM4 output used as CLC3 Input B
– Signal 3 (Blue) is CLC3 output (logic XOR between PWM3 and PWM4)

Figure 2-4. Oscillograms of CLC3 Performing Logic XOR, Input Signals PWM3 and PWM4



To achieve the functionality described by the use case, the following actions will have to be performed:

- System clock initialization
- Port initialization
- Timer initialization
- PWM initialization
- CLC initialization.

2.1 MCC Generated Code

To generate this project using MPLAB® Code Configurator (MCC), follow the next steps:

1. Create a new MPLAB X IDE project for PIC18F47Q10.
2. Open MCC from the toolbar. More information on how to install the MCC plug-in can be found here.
3. Go to *Project Resources → System → System Module* and do the following configuration:
   – Oscillator Select: HFINTOSC
   – HF Internal Clock: 64 MHz
   – Clock Divider: 1
   – In the Watchdog Timer Enable field in the **WWDT** tab, **WDT Disabled** has to be selected.
   – In the **Programming** tab, **Low-Voltage Programming Enable** has to be checked.
4. From the Device Resources window, add TMR2, TMR4, PWM3, PWM4, CLC1, CLC2 and CLC3. Do the following configurations for each peripheral:

   4.1. **Timer2 Configuration:**
   - Enable Timer: checked
   - **Timer Clock** tab
     – Clock Source: FOSC/4
     – Prescaler: 1:1

  - Postscaler: 1:1
- Timer Period: 10 us
- Enable Timer Interrupt: unchecked

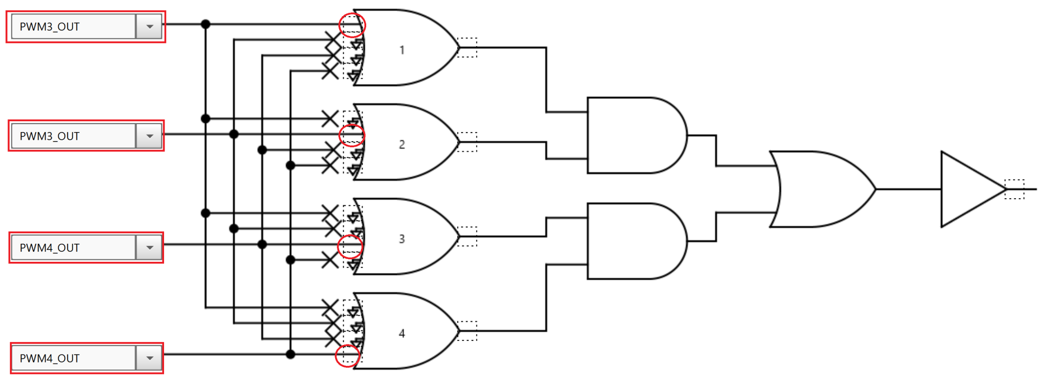
4.2. **Timer4 Configuration:**
- Enable Timer: checked
- Ext Reset Source: TMR2_postscaled
- Start/Reset Options: Resets at rising TMR4_rst
- **Timer Clock** tab
  - Clock Source: FOSC/4
  - Prescaler: 1:1
  - Postscaler: 1:1
- Timer Period: 5 us
- Enable Timer Interrupt: unchecked

4.3. **PWM3 Configuration:**
- Enable PWM: checked
- Select a Timer: Timer2
- Duty Cycle: 50%
- PWM Polarity: active_hi

4.4. **PWM4 Configuration:**
- Enable PWM: checked
- Select a Timer: Timer4
- Duty Cycle: 50%
- PWM Polarity: active_hi

4.5. **CLC1 Configuration:**
- Enable CLC: checked
- Mode: AND-OR
- The AND-OR user configurable interpretation window is now available in the CLC1 window. Set the internal connections as shown below. By selecting **PWM3** as both inputs for the AND, it becomes the first input of the OR. The same is true for **PWM4** as second input of OR; this way CLC1 is used as a 2-input OR gate.

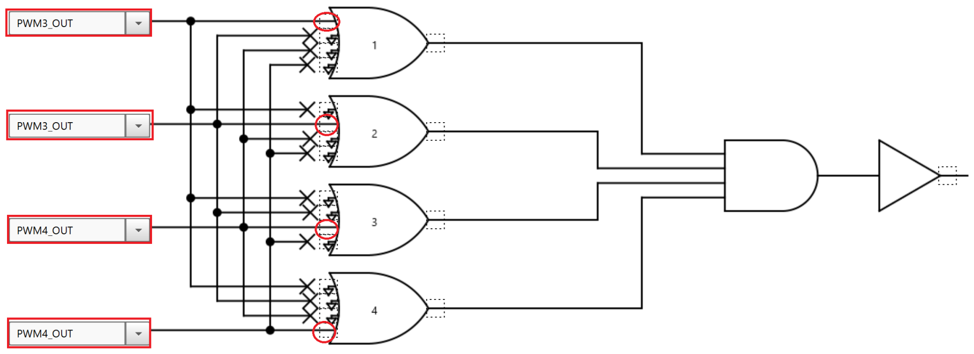
Figure 2-5. CLC1 Configuration Mode AND-OR, Used as 2-Input OR



4.6. **CLC2 Configuration:**

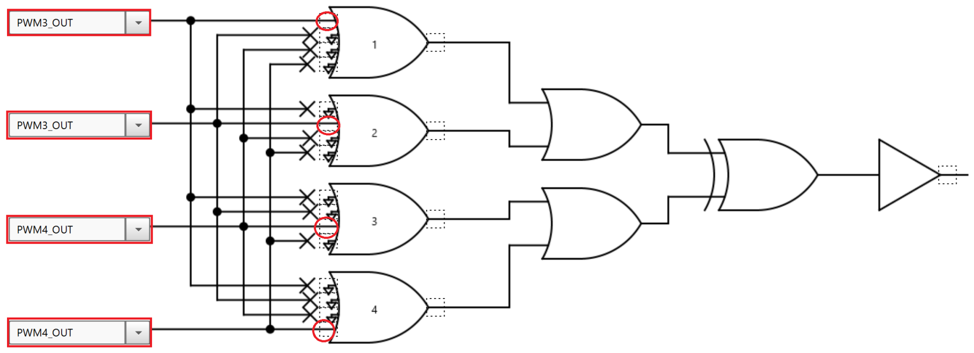
- Enable CLC: checked
- Mode: 4-input AND
- The 4-input AND user configurable interpretation window is now available in the CLC2 window. Set the internal connections as shown below. By selecting **PWM3** as two of the inputs for the 4-input AND, and **PWM4** as the other two, it becomes equivalent to a 2-input AND function from CLC2.

Figure 2-6. CLC2 Configuration Mode 4-Input AND, Used as 2-Input AND



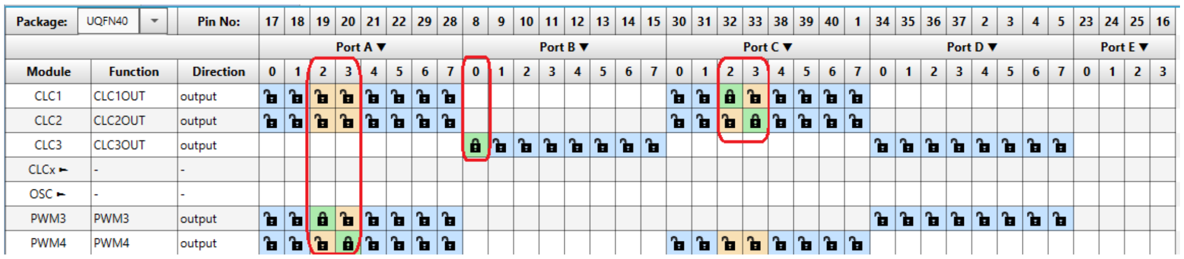
4.7. **CLC3 Configuration:**
- Enable CLC: checked
- Mode: 4-input OR-XOR
- The OR-XOR user configurable interpretation window is now available in the CLC3 window. Set the internal connections as shown below. By selecting **PWM3** as both inputs for the OR, it becomes the first input of the XO. The same is true for **PWM4** as second input of XOR; this way CLC3 is used as a 2-input XOR gate.

Figure 2-7. CLC3 Configuration Mode OR-XOR, Used as 2-Input XOR



5. In the Pin Manager: Grid View window, select the I/O pins outputs to enable the internal signal access to the I/O as shown in Figure 2-8.

Figure 2-8. Pin Mapping for the Example “Using Basic Logic Gates”



Package:	UQFN40	Pin No:	17	18	19	20	21	22	29	28	8	9	10	11	12	13	14	15	30	31	32	33	38	39	40	1	34	35	36	37	2	3	4	5	23	24	25	16
			Port A ▼								Port B ▼								Port C ▼								Port D ▼								Port E ▼			
Module	**Function**	**Direction**	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3
CLC1	CLC1OUT	output	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓									🔓	🔓	🔒	🔓	🔓	🔓	🔓	🔓												
CLC2	CLC2OUT	output	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓									🔓	🔓	🔓	🔒	🔓	🔓	🔓	🔓												
CLC3	CLC3OUT	output									🔒	🔓	🔓	🔓	🔓	🔓	🔓	🔓									🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓				
CLCx ►	-	-																																				
OSC ►	-	-																																				
PWM3	PWM3	output	🔓	🔓	🔒	🔓	🔓	🔓	🔓	🔓																	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓				
PWM4	PWM4	output	🔓	🔓	🔓	🔒	🔓	🔓	🔓	🔓									🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓												

6. In the Project Resources window, click **Generate**.

For this example, no extra code was used aside from the one generated from MCC.

View the PIC18F47Q10 Code Example on GitHub
Click to browse repositories

2.2 Bare Metal Code

The necessary code and functions to implement the presented example are analyzed in this section.

The first step will be to configure the microcontroller to disable the Watchdog Timer and to enable the Low-Voltage Programming (LVP).

```
#pragma config WDTE = OFF     /* WDT operating mode → WDT Disabled */
#pragma config LVP = ON       /* Low-voltage programming enabled, RE3 pin is MCLR */
```

As described in the example functionality, the following peripherals must be initialized: Timer2, Timer4, PWM3, PWM4, CLC1, CLC2, CLC3, the I/O PORT and PPS.

The internal oscillator has to be set to the desired value (in this case to 64 MHz), using the following function:

```
static void CLK_init(void)
{
    OSCCON1bits.NOSC = 6;   /* HFINTOSC oscillator */
    OSCFRQ = 0x08;          /* HFFRQ 64_MHz; */
}
```

To enable the output driver in the desired I/O pins (RA2, RA3, RB0, RC2 and RC3), the following function is used:

```
static void PORT_init(void)
{
    /* PORT RA2 and RA3 output driver enabled */
    TRISAbits.TRISA2 = 0;
    TRISAbits.TRISA3 = 0;
    /* PORT RB0 output driver enabled */
    TRISBbits.TRISB0 = 0;
    /* PORT RC2 and RC3 output driver enabled */
    TRISCbits.TRISC2 = 0;
    TRISCbits.TRISC3 = 0;
}
```

For Timer2 to use FOSC/4 as clock source and generate a pulse every 10 us (frequency = 100 kHz), the following function is used:

```
static void TMR2_init(void)
{
    T2CLKCONbits.CS = 1;  /* Timer2 clock source is FOSC/4 */
    T2PR = 0x9F;          /* Load period values */
    T2CON = 0x80;         /* Enable Timer2 */
}
```

Timer4 is configured to use FOSC/4 as clock source and generate a pulse every 5 us (frequency = 200 kHz). Timer4 is also set to start at the same time as Timer2. The following function is used:

```
static void TMR4_init(void)
{
    /* Timer4 clock source is FOSC/4 */
    T4CLKCONbits.CS = 1;
    /* Timer4 resets at rising TMR4_ers*/
    T4HLTbits.MODE = 4;
    /* Timer4 Reset source is TMR2_postscaled; */
    T4RSTbits.RSEL = 1;
    /* Load period values */
    T4PR = 0x4F;
    /* Enable Timer4 */
    T4CON = 0x80;
}
```

PWM3 uses Timer2 as pulse source and is configured to generate a pulse with 50% duty cycle. The following function is used:

```
static void PWM3_init(void)
 {
    PWM3CON = 0x80;   /* Enable PWM3*/
    /* Load duty cycle values */
    PWM3DCH = 0x4F;
    PWM3DCL = 0xC0;
    CCPTMRS = 0x10;   /* Select Timer2 as pulse source */
 }
```

PWM4 uses Timer4 as pulse source and it is configured to generate a pulse with 50% duty cycle. The following function is used:

```
static void PWM4_init(void)
 {
    PWM4CON = 0x80;           /* Enable PWM4*/
    /* Load duty cycle values */
    PWM4DCH = 0x27;
    PWM4DCL = 0xC0;
    CCPTMRSbits.P4TSEL = 2;   /*  Select Timer4 as pulse source */
 }
```

CLC1 is configured in the AND-OR mode and uses PWM3 and PWM4 as inputs with the instructions from the following function:

```
static void CLC1_init(void)
{
    CLC1POL = 0x00;    /* Clear the output polarity register */
    CLC1SEL0 = 0x1A;   /* Configure PWM3_OUT as input for first OR gate */
    CLC1SEL1 = 0x1A;   /* Configure PWM3_OUT as input for second OR gate */
    CLC1SEL2 = 0x1B;   /* Configure PWM4_OUT as input for third OR gate */
    CLC1SEL3 = 0x1B;   /* Configure PWM4_OUT as input for forth OR gate */
    /* All four inputs are not inverted*/
    CLC1GLS0 = 0x02;
    CLC1GLS1 = 0x08;
    CLC1GLS2 = 0x20;
    CLC1GLS3 = 0x80;
    CLC1CON = 0x80;    /* CLC1 enabled; Mode AND-OR*/
}
```

CLC2 is configured in the 4-input AND mode and uses PWM3 and PWM4 as inputs with the instructions from the following function:

```
static void CLC2_init(void)
{
    CLC2POL = 0x00;    /* Clear the output polarity register */
    CLC2SEL0 = 0x1A;   /* Configure PWM3_OUT as input for first OR gate */
    CLC2SEL1 = 0x1A;   /* Configure PWM3_OUT as input for second OR gate */
    CLC2SEL2 = 0x1B;   /* Configure PWM4_OUT as input for third OR gate */
    CLC2SEL3 = 0x1B;   /* Configure PWM4_OUT as input for forth OR gate */
    /* All four inputs are not inverted*/
    CLC2GLS0 = 0x02;
    CLC2GLS1 = 0x08;
    CLC2GLS2 = 0x20;
    CLC2GLS3 = 0x80;
    CLC2CONbits.EN = 1;     /* CLC2 enabled */
    CLC2CONbits.MODE = 2;   /* Mode 4-input AND */
}
```

CLC3 is configured in the OR-XOR mode and uses PWM3 and PWM4 as inputs with the instructions from the following function:

```
static void CLC3_init(void)
{
    CLC3POL = 0x00;    /* Clear the output polarity register */
    CLC3SEL0 = 0x1A;   /* Configure PWM3_OUT as input for first OR gate */
    CLC3SEL1 = 0x1A;   /* Configure PWM3_OUT as input for second OR gate */
    CLC3SEL2 = 0x1B;   /* Configure PWM4_OUT as input for third OR gate */
    CLC3SEL3 = 0x1B;   /* Configure PWM4_OUT as input for forth OR gate */
    /* All four inputs are not inverted*/
    CLC3GLS0 = 0x02;
    CLC3GLS1 = 0x08;
    CLC3GLS2 = 0x20;
    CLC3GLS3 = 0x80;
    CLC3CONbits.EN = 1;     /* CLC3 enabled */
    CLC3CONbits.MODE = 1;   /* Mode OR-XOR */
}
```

To measure the internal peripheral signals with the oscilloscope, the following link must be made:

Table 2-4. Peripheral Mapping to I/O Pins for the Example "Using Basic Logic Gates"

Internal CIP Signal	Microcontroller Pin
PWM3_OUT	RA2
PWM4_OUT	RA3
CLC1_OUT	RC2
CLC2_OUT	RC3
CLC3_OUT	RB0

This is done in the following function:

```
static void PPS_init(void)
{
    RA2PPS = 0x07;   /*Configure RA2 for PWM3 output*/
    RA3PPS = 0x08;   /*Configure RA3 for PWM4 output*/
    RB0PPS = 0x1A;   /*Configure RB0 for CLC3 output*/
    RC2PPS = 0x18;   /*Configure RC2 for CLC1 output*/
    RC3PPS = 0x19;   /*Configure RC3 for CLC2 output*/
}
```

View the PIC18F47Q10 Code Example on GitHub
Click to browse repositories

3. Using CLCs to Create a Data Signal Modulator (DSM)

This example shows an initialization of the CLC in the J-K flip-flop with R mode and AND-OR mode for the implementation of a Data Signal Modulator (DSM) with timings controlled from the CCP peripheral. The truth table of the J-K flip-flop is shown below.

Table 3-1. Truth Table for the J-K Flip-Flop

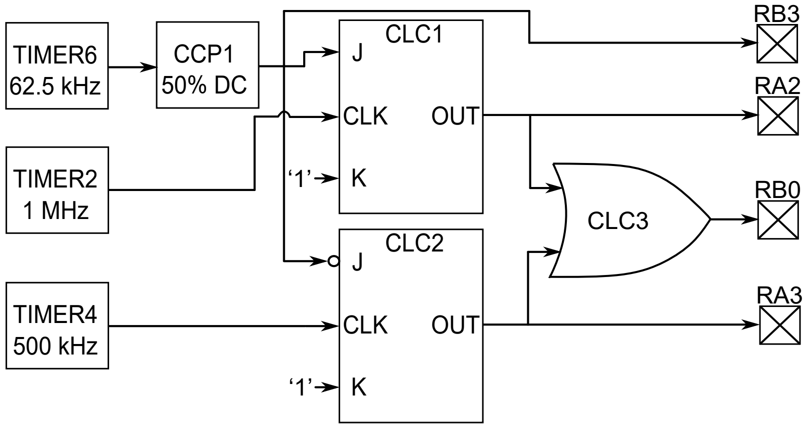
CLK	J	K	O
0	0	0	Latch
0	0	1	0
0	1	0	1
0	1	1	Toggle
1	0	0	Latch
1	0	1	Latch
1	1	0	Latch
1	1	1	Latch

CLC1 (J-K flip-flop 1) has Timer2 as clock source (which represents the first modulated frequency), while the carrier signal generated from the CCP is connected to the J gate of the J-K flip-flop. The K gate is left to logic 1. This allows the CLC to toggle when the J input is high and stay 0 when the J input is low.

CLC2 (J-K flip-flop 2) is connected in the same way with Timer4 as clock source and the negated CCP as input for the J gate. This CCP connection to the J gates ensures that one CLC is toggling while the other has the output set to 0 logic. CLC3 set in AND-OR mode is connecting CLC1 and CLC2 outputs to create a DSM.

The internal architecture is shown in Figure 3-1.

Figure 3-1. Internal Connections for "Using CLCs to Create a DSM" Example



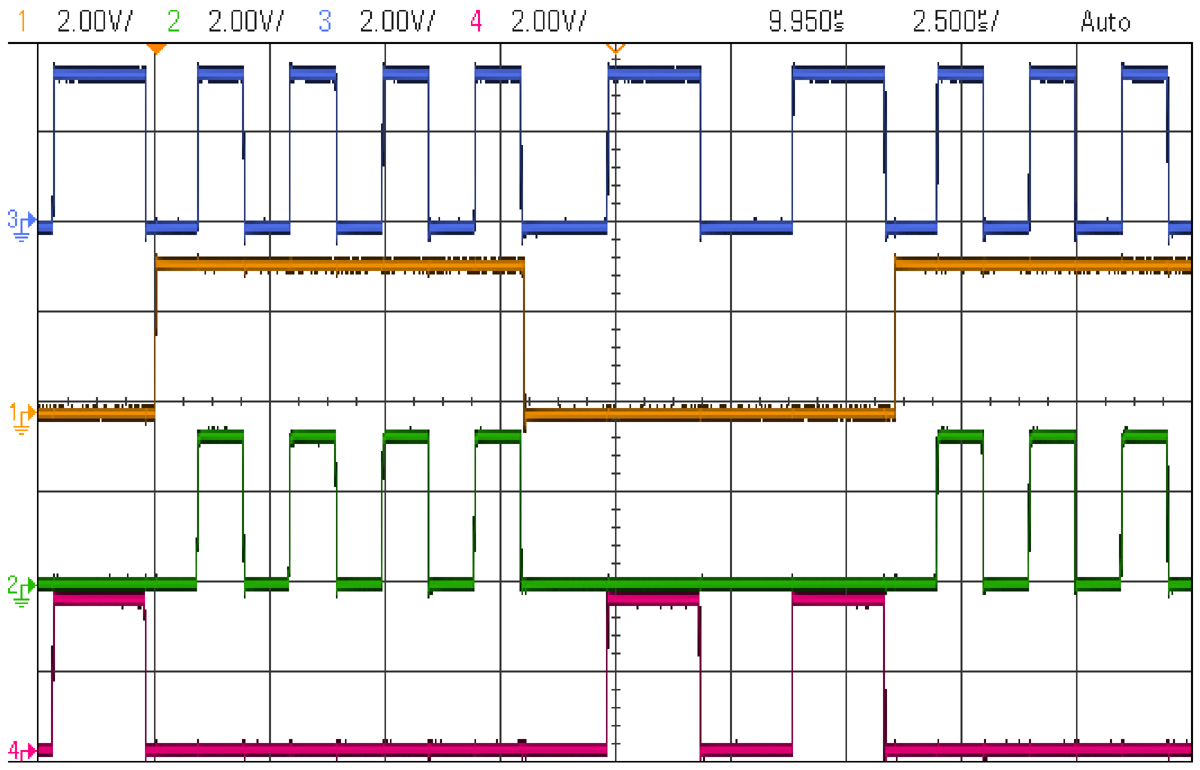
DSM setup configurations:

- Timer2 frequency = 1 MHz (1 us period)
- Timer4 frequency = 500 kHz (2 us period)
- Timer6 frequency = 62.5 kHz (16 us period)
- CCP1 has Timer6 as source and duty cycle = 50%
- CLC1 is set up as J-K flip-flop with R
- CLC2 is set up as J-K flip-flop with R
- CLC3 is set up as AND-OR: used as 2-input OR

Figure 3-2 displays all the CLCs outputs and the CCP1 output side by side to show how this configuration implements the DSM function:

- Signal 1 (Orange) is CCP1 output
- Signal 2 (Green) is CLC1 output
- Signal 3 (Blue) is CLC3 output
- Signal 4 (Red) is CLC2 output

Figure 3-2. Oscillograms of CLC1, CLC2, CLC3 and CCP1 Performing a DSM Function



To achieve the functionality described by the use case, the following actions will have to be performed:

- System clock initialization
- Port initialization
- Timer initialization
- CCP initialization
- CLC initialization.

3.1 MCC Generated Code

To generate this project using MPLAB® Code Configurator (MCC), follow the next steps:

1. Create a new MPLAB X IDE project for PIC18F47Q10.
2. Open MCC from the toolbar. More information on how to install the MCC plug-in can be found here.
3. Go to *Project Resources → System → System Module* and do the following configuration:
   – Oscillator Select: HFINTOSC
   – HF Internal Clock: 64 MHz
   – Clock Divider: 1
   – In the Watchdog Timer Enable field in the **WWDT** tab, **WDT Disabled** has to be selected.
   – In the **Programming** tab, **Low-Voltage Programming Enable** has to be checked.
4. From the Device Resources window, add TMR2, TMR4, TMR6, CCP1, CLC1, CLC2 and CLC3. Do the following configurations for each peripheral:

4.1. **Timer2 Configuration:**
- Enable Timer: checked
- **Timer Clock** tab
  – Clock Source: FOSC/4
  – Prescaler: 1:1
  – Postscaler: 1:1
- Timer Period: 1 us
- Enable Timer Interrupt: unchecked

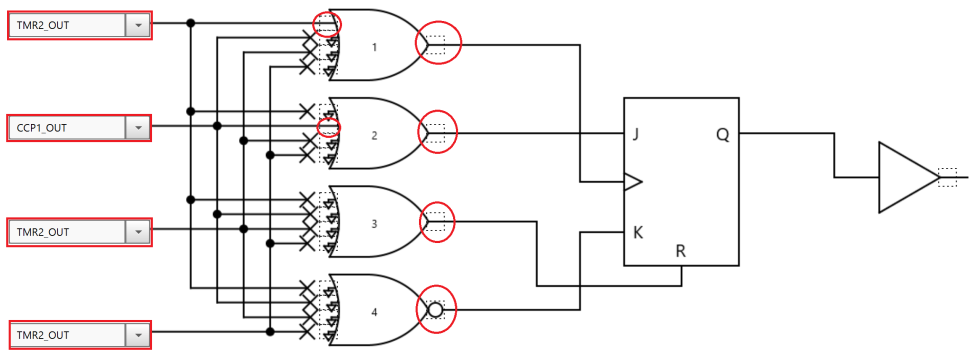
4.2. **Timer4 Configuration:**
- Enable Timer: checked
- **Timer Clock** tab
  – Clock Source: FOSC/4
  – Prescaler: 1:1
  – Postscaler: 1:1
- Timer Period: 2 us
- Enable Timer Interrupt: unchecked

4.3. **Timer6 Configuration:**
- Enable Timer: checked
- **Timer Clock** tab
  – Clock Source: FOSC/4
  – Prescaler: 1:1
  – Postscaler: 1:1
- Timer Period: 16 us
- Enable Timer Interrupt: unchecked

4.4. **CCP1 Configuration:**
- Enable CCP: checked
- CCP Mode: PWM
- Select Timer: Timer6
- Duty Cycle: 50%
- CCPR Alignment: right_aligned

4.5. **CLC1 Configuration:**
- Enable CLC: checked
- Mode: J-K flip-flop with R
- The J-K flip-flop with R user configurable interpretation window is now available in the CLC1 window. Set the internal connections as shown below. By selecting **TMR2** as clock source for the J-K, the CLC will generate a PWM signal with 500 kHz frequency and 50% duty cycle. Select **CCP1** as the J input, allowing to have O = 0 when J is 0 and K is 1.

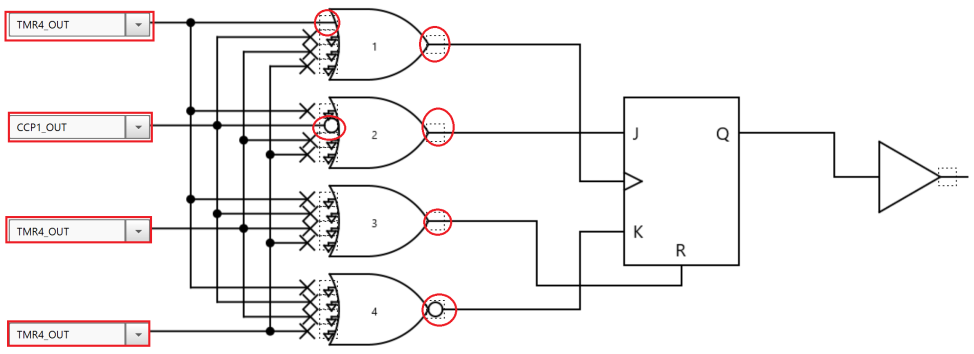
Figure 3-3. CLC1 Configuration Mode J-K Flip-Flop with R



4.6. CLC2 Configuration:

- Enable CLC: checked
- Mode: J-K flip-flop with R
- The J-K flip-flop with R user configurable interpretation window is now available in the CLC2 window. Set the internal connections as shown below. By selecting **TMR** as clock source for the J-K, the CLC will generate a PWM signal with 250 kHz frequency and 50% duty cycle. Select **CCP1** as the J input, allowing to have O = 0 when J is 0 and K is 1.

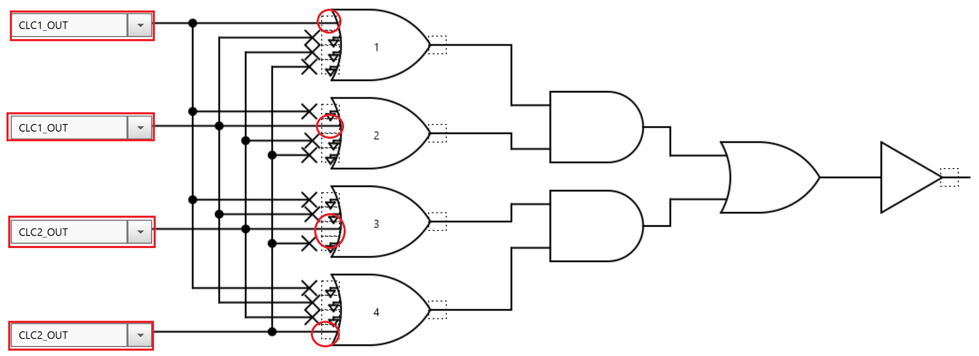
Figure 3-4. CLC2 Configuration Mode J-K Flip-Flop with R



4.7. CLC3 Configuration:

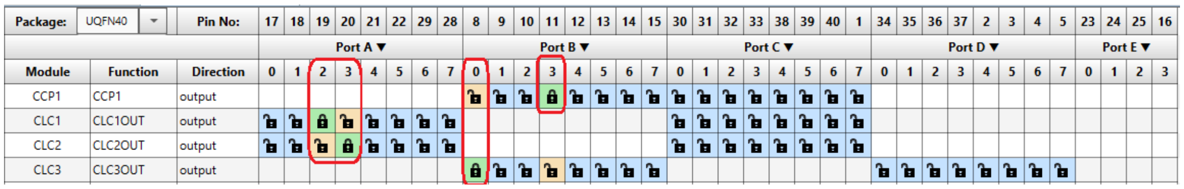
- Enable CLC: checked
- Mode: AND-OR
- The AND-OR user configurable interpretation window is now available in the CLC3 window. Set the internal connections as shown below. By selecting **CLC1** as both inputs for the AND, it becomes the first input of the OR. The same is true for **CLC2** as second input of OR; this way CLC3 is used as a 2-input OR gate.

Figure 3-5. CLC3 Configuration Mode AND-OR, Used as 2-Input OR



5. In the Pin Manager: Grid View window, select the I/O pins outputs to enable the internal signal access to the I/O as shown in Figure 3-6.

Figure 3-6. Pin Mapping for the Example "Using CLCs to Create a Data Signal Modulator"



Package: UQFN40		Pin No:	17	18	19	20	21	22	29	28	8	9	10	11	12	13	14	15
			Port A ▼								Port B ▼							
Module	Function	Direction	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
CCP1	CCP1	output									🔓	🔓	🔓	🔒	🔓	🔓	🔓	🔓
CLC1	CLC1OUT	output	🔓	🔓	🔒	🔓	🔓	🔓	🔓	🔓								
CLC2	CLC2OUT	output	🔓	🔓	🔓	🔒	🔓	🔓	🔓	🔓								
CLC3	CLC3OUT	output									🔒	🔓	🔓	🔓	🔓	🔓	🔓	🔓

Package: UQFN40		Pin No:	30	31	32	33	38	39	40	1	34	35	36	37	2	3	4	5	23	24	25	16
			Port C ▼								Port D ▼								Port E ▼			
Module	Function	Direction	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3
CCP1	CCP1	output	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓												
CLC1	CLC1OUT	output	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓												
CLC2	CLC2OUT	output	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓												
CLC3	CLC3OUT	output									🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓				

6. In the Project Resources window, click **Generate**.

For this example, no extra code was used aside from the one generated from MCC.

View the PIC18F47Q10 Code Example on GitHub
Click to browse repositories

3.2 Bare Metal Code

The necessary code and functions to implement the presented example are analyzed in this section.

The first step will be to configure the microcontroller to disable the Watchdog Timer and to enable the Low-Voltage Programming (LVP).

```
#pragma config WDTE = OFF     /* WDT operating mode → WDT Disabled */
#pragma config LVP = ON       /* Low-voltage programming enabled, RE3 pin is MCLR */
```

As described in the example functionality, the following peripherals must be initialized: Timer2, Timer4, Timer6, CCP1, CLC1, CLC2, CLC3, the I/O PORT and PPS.

The internal oscillator has to be set to the desired value (in this case to 64 MHz), using the following function:

```
static void CLK_init(void)
{
    OSCCON1bits.NOSC = 6;    /* HFINTOSC oscillator */
    OSCFRQ = 0x08;           /* HFFRQ 64_MHz; */
}
```

To enable the output driver in the desired I/O pins (RA2, RA3 and RB3), the following function is used:

```
static void PORT_init(void)
{
    /*PORT RA2 and RA3 output driver enabled*/
    TRISAbits.TRISA2 = 0;
    TRISAbits.TRISA3 = 0;
    /*PORT RB3 and RB0 output driver enabled*/
    TRISBbits.TRISB0 = 0;
    TRISBbits.TRISB3 = 0;
}
```

Timer2 is configured to use FOSC/4 as clock source and generate a pulse every 1 us (frequency = 1 MHz). Timer2 is also set to start at the same time as Timer4. The following function is used:

```
static void TMR2_init(void)
{
    T2CLKCONbits.CS = 1;   /* Timer2 clock source is FOSC/4 */
    T2HLTbits.MODE = 4;    /* Timer2 resets at rising TMR2_ers*/
    T2RSTbits.RSEL = 2;    /* Timer2 Reset source is TMR4_postscaled; */
    T2PR = 0x0F;           /* Load period values */
    T2CON = 0x80;          /* Enable Timer2 */
}
```

Timer4 is configured to use FOSC/4 as clock source and generate a pulse every 2 us (frequency = 200 kHz). The following function is used:

```
static void TMR4_init(void)
{
    T4CLKCONbits.CS = 1;    /* Timer4 clock source is FOSC/4 */
    T4PR = 0x1F;            /* Load period values */
    T4CON = 0x80;           /* Enable Timer4 */
}
```

Timer6 is configured to use FOSC/4 as clock source and generate a pulse every 16 us (frequency = 62.5 kHz). The following function is used:

```
static void TMR6_init(void)
{
    T6CLKCONbits.CS = 1;    /* Timer6 clock source is FOSC/4 */
    T6PR = 0xFF;            /* Load period values */
    T6CON = 0x80;           /* Enable Timer6 */
}
```

CCP1 works as a PWM with 50% duty cycle and Timer6 as pulse source. The following function is used:

```
static void CCP1_init(void)
 {
    CCP1CON = 0x8C;    /* Enable CCP1 in PWM mode*/
    /* Load duty cycle values */
    CCPR1H = 0x01;
    CCPR1L = 0xFF;
    CCPTMRS = 0x03;    /* Select Timer6 as pulse source*/
 }
```

CLC1 is configured in the J-K flip-flop with R mode and uses TMR2 and CCP1 as inputs. The following function is used:

```
static void CLC1_init(void)
{
    CLC1POL = 0x08;     /* Negated output for fourth OR gate*/
    CLC1SEL0 = 0x13;    /* Configure TMR2_OUT as input for first OR gate */
    CLC1SEL1 = 0x18;    /* Configure CCP1_OUT as input for second OR gate */
    CLC1SEL2 = 0x13;    /* Configure TMR2_OUT as input for third OR gate */
    CLC1SEL3 = 0x13;    /* Configure TMR2_OUT as input for forth OR gate */
    /* All four inputs are not inverted*/
    CLC1GLS0 = 0x02;
    CLC1GLS1 = 0x08;
    CLC1GLS2 = 0x00;
    CLC1GLS3 = 0x00;
    CLC1CONbits.EN = 1;     /* CLC1 enabled */
    CLC1CONbits.MODE = 6;   /* Mode J-K flip-flop with R */
}
```

CLC2 is configured in the J-K flip-flop with R mode and uses TMR4 and CCP1 as inputs. The following function is used:

```
static void CLC2_init(void)
{
    CLC2POL = 0x08;    /* Negated output for fourth OR gate*/
    CLC2SEL0 = 0x15;   /* Configure TMR4_OUT as input for first OR gate */
    CLC2SEL1 = 0x18;   /* Configure CCP1_OUT as input for second OR gate */
    CLC2SEL2 = 0x15;   /* Configure TMR4_OUT as input for third OR gate */
    CLC2SEL3 = 0x15;   /* Configure TMR4_OUT as input for forth OR gate */
    /* Inputs 1, 3 and 4 are not inverted; Input 2 inverted*/
    CLC2GLS0 = 0x02;
    CLC2GLS1 = 0x04;
    CLC2GLS2 = 0x00;
    CLC2GLS3 = 0x00;
    CLC2CONbits.EN = 1;     /* CLC2 enabled */
    CLC2CONbits.MODE = 6;   /* Mode J-K flip-flop with R */
}
```

CLC3 is configured in the AND-OR mode and uses CLC1 and CLC2 as inputs. The following function is used:

```
static void CLC3_init(void)
{
    CLC3POL = 0x00;    /* Clear the output polarity register */
    CLC3SEL0 = 0x21;   /* Configure CLC1_OUT as input for first OR gate */
    CLC3SEL1 = 0x21;   /* Configure CLC1_OUT as input for second OR gate */
    CLC3SEL2 = 0x22;   /* Configure CLC2_OUT as input for third OR gate */
    CLC3SEL3 = 0x22;   /* Configure CLC2_OUT as input for forth OR gate */
    /* All four inputs are not inverted*/
    CLC3GLS0 = 0x02;
    CLC3GLS1 = 0x08;
    CLC3GLS2 = 0x20;
    CLC3GLS3 = 0x80;
    CLC3CONbits.EN = 1;     /* CLC3 enabled */
    CLC3CONbits.MODE = 0;   /* Mode AND-OR */
}
```

To measure the internal peripheral signals with the oscilloscope, the following link must be made:

Table 3-2. Peripheral Mapping to I/O Pins for the Example "Using CLCs to Create a Data Signal Modulator"

Internal CIP Signal	Microcontroller Pin
CLC1_OUT	RA2
CLC2_OUT	RA3
CLC3_OUT	RB0
CCP1_OUT	RB3

This is done in the following function:

```
static void PPS_init(void)
{
    RA2PPS = 0x18;   /*Configure RA2 for CLC1 output*/
    RA3PPS = 0x19;   /*Configure RA3 for CLC2 output*/
    RB0PPS = 0x1A;   /*Configure RB0 for CLC3 output*/
    RB3PPS = 0x05;   /*Configure RB3 for CCP1 output*/
}
```

View the PIC18F47Q10 Code Example on GitHub
Click to browse repositories

4. Using the CLC to Create an LED Dimming Effect

This example shows an initialization of the CLC in the SR Latch mode for the implementation of an automatic fixed frequency with variable duty cycle PWM signal. The function is automatically called as it does not require code or core supervision to work. The hardware is set at start-up and can be reconfigured during run-time. The truth table of the SR Latch is shown below.

Table 4-1. Truth Table for the SR Latch

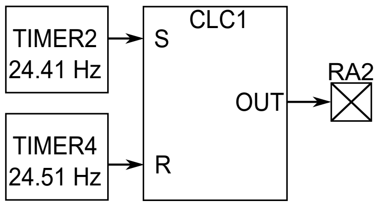
S	R	O
0	0	0
0	1	0
1	0	1
1	1	0

CLC1 (SR Latch) has Timer2 as S gate source (which creates the fixed frequency of the generated PWM), and connected at the R gate is Timer4 to generate the duty cycle of the PWM. Timer4 is set with a slightly higher frequency than Timer2. At each cycle, the duty cycle will get smaller with the difference between the two signals until it reaches zero and the process restarts. The difference between Timer2 and Timer4 represents the step of the duty cycle change, which in this case is 0.1 Hz (or 1 Least Significant Byte difference), and therefore creating 255 repetitive PWM signals.

When this PWM signal is connected to an LED, the created effect is that the LED is repetitively dimmed. If the signal is inversed, it creates the effect of repetitively increasing LED light (the kind of loading effect).

The internal architecture is shown in Figure 4-1.

Figure 4-1. Internal Connections for "Using CLC to Create an LED Dimming Effect" Example



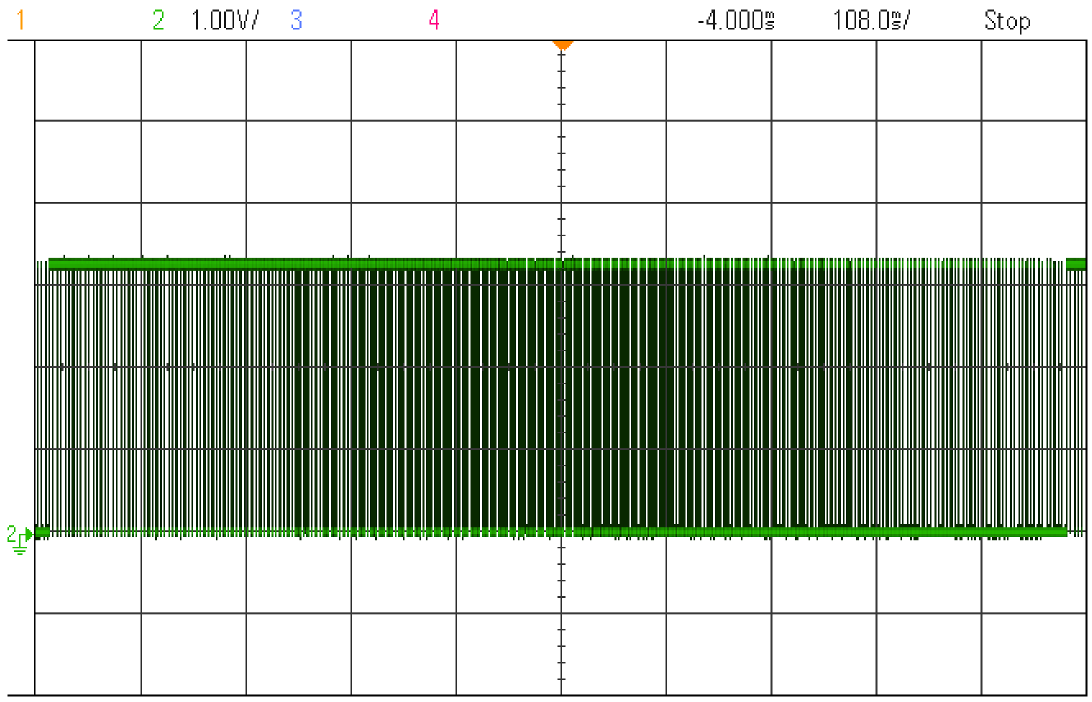
Test setup configurations:

- Timer2 frequency = 24.41 Hz (4.096 ms period)
- Timer4 frequency = 24.51 Hz (4.08 ms period)
- CLC1 is set up as SR Latch

Figure 4-2 displays the CLC1 output implementing the fixed frequency with variable duty cycle function:

- Signal 2 (Green) is CLC1 output

Figure 4-2. Oscillograms of CLC1 Output Generating the Fixed Frequency Variable Duty Cycle PWM Signal



To achieve the functionality described by the use case, the following actions will have to be performed:

- System clock initialization
- Port initialization
- Timer initialization
- CLC initialization.

4.1 MCC Generated Code

To generate this project using MPLAB® Code Configurator (MCC), follow the next steps:

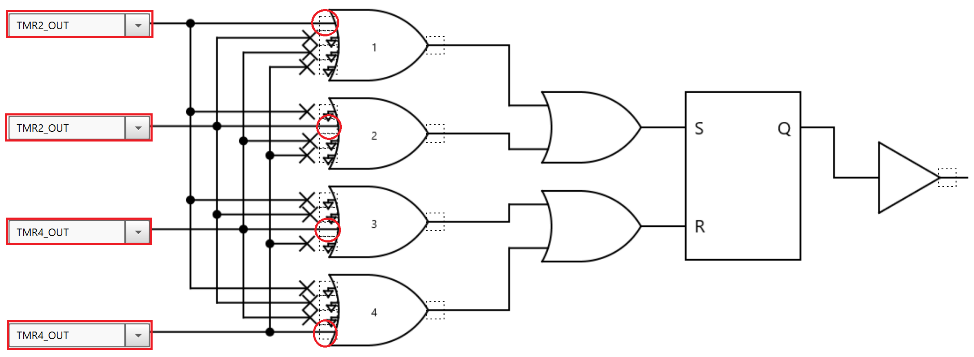
1. Create a new MPLAB X IDE project for PIC18F47Q10.
2. Open MCC from the toolbar. More information on how to install the MCC plug-in can be found here.
3. Go to *Project Resources → System → System Module* and do the following configuration:
   – Oscillator Select: HFINTOSC
   – HF Internal Clock: 64 MHz
   – Clock Divider: 1
   – In the Watchdog Timer Enable field in the **WWDT** tab, WDT Disabled has to be selected.
   – In the **Programming** tab, **Low-Voltage Programming Enable** has to be checked.
4. From the Device Resources window, add TMR2, TMR4 and CLC1. Do the following configurations for each peripheral:

   4.1. **Timer2 Configuration:**
   - Enable Timer: checked
   - **Timer Clock** tab
     – Clock Source: FOSC/4
     – Prescaler: 1:128
     – Postscaler: 1:2
   - Timer Period: 4.096 ms
   - Enable Timer Interrupt: unchecked

4.2. **Timer4 Configuration:**
- Enable Timer: checked
- **Timer Clock** tab
  - Clock Source: FOSC/4
  - Prescaler: 1:128
  - Postscaler: 1:2
- Timer Period: 4.08 ms
- Enable Timer Interrupt: unchecked

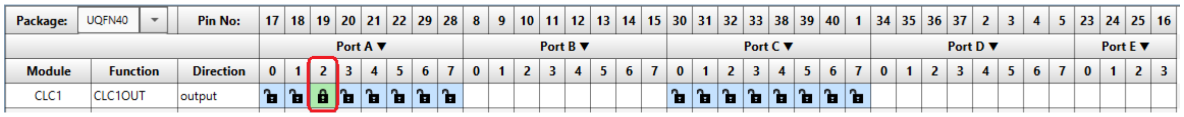
4.3. **CLC1 Configuration:**
- Enable CLC: checked
- Mode: SR Latch
- The SR Latch user configurable interpretation window is now available in the CLC1 window. Set the internal connections as shown below. By selecting **TMR2** as clock source for the 'S' input and **TMR4** as source for the 'R' input, the CLC will generate a PWM signal with 24.51 Hz frequency and variable duty cycle with steps equal to the difference between TMR2 and TMR4.

Figure 4-3. CLC1 Configuration Mode SR Latch



5. In the Pin Manager: Grid View window, select the I/O pins outputs to enable the internal signal access to the I/O as shown in Figure 4-4.

Figure 4-4. Pin Mapping for the Example "Using the CLC to Create an LED Dimming Effect"



Package:	UQFN40	Pin No:	17	18	19	20	21	22	29	28	8	9	10	11	12	13	14	15	30	31	32	33	38	39	40	1	34	35	36	37	2	3	4	5	23	24	25	16
			Port A ▼								Port B ▼								Port C ▼								Port D ▼								Port E ▼			
Module	Function	Direction	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3
CLC1	CLC1OUT	output	🔓	🔓	🔒	🔓	🔓	🔓	🔓	🔓									🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓												

6. In the Project Resources window, click **Generate**.

For this example, no extra code was used aside from the one generated from MCC.

View the PIC18F47Q10 Code Example on GitHub
Click to browse repositories

4.2 Bare Metal Code

The necessary code and functions to implement the presented example are analyzed in this section.

The first step will be to configure the microcontroller to disable the Watchdog Timer and to enable the Low-Voltage Programming (LVP).

```
#pragma config WDTE = OFF      /* WDT operating mode → WDT Disabled */
#pragma config LVP = ON        /* Low-voltage programming enabled, RE3 pin is MCLR */
```

As described in the example functionality, the following peripherals must be initialized: Timer2, Timer4, CLC1, the I/O PORT and PPS.

The internal oscillator has to be set to the desired value (in this case to 64 MHz), using the following function:

```
static void CLK_init(void)
{
    OSCCON1bits.NOSC = 6;    /* HFINTOSC oscillator */
    OSCFRQ = 0x08;           /* HFFRQ 64_MHz; */
}
```

To enable the output driver in the desired I/O pins (RA2), the following function is used:

```
static void PORT_init(void)
{
    TRISAbits.TRISA2 = 0;    /*PORT RA2 output driver enabled*/
}
```

For Timer2 to use FOSC/4 as clock source and generate a pulse every 4.096 ms (frequency = 24.41 Hz), the prescaller must also be added to 1:128 and postcaller to 1:2. The following function is used:

```
static void TMR2_init(void)
{
    T2CLKCONbits.CS = 1;    /* Timer2 clock source is FOSC/4 */
    T2PR = 0xFF;            /* Load period values */
    T2CONbits.CKPS = 7;     /* Set prescaller to 1:128 */
    T2CONbits.OUTPS = 1;    /* Set postcaller to 1:2 */
    T2CONbits.ON = 1;       /* Enable Timer2 */
}
```

For Timer4 to use FOSC/4 as clock source and generate a pulse every 4.08 ms (frequency = 24.51 Hz), the prescaller must also be added to 1:128 and postcaller to 1:2. The following function is used:

```
static void TMR4_init(void)
{
    T4CLKCONbits.CS = 1;    /* Timer4 clock source is FOSC/4 */
    T4PR = 0xFE;            /* Load period values */
    T4CONbits.CKPS = 7;     /* Set prescaller to 1:128 */
    T4CONbits.OUTPS = 1;    /* Set postcaller to 1:2 */
    T4CONbits.ON = 1;       /* Enable Timer4 */
}
```

CLC1 is configured in the SR Latch mode and uses TMR2 and TMR4 as inputs. The following function is used:

```
static void CLC1_init(void)
{
    CLC1POL = 0x00;     /* Clear the output polarity register */
    CLC1SEL0 = 0x13;    /* Configure TMR2_OUT as input for first OR gate */
    CLC1SEL1 = 0x13;    /* Configure TMR2_OUT as input for second OR gate */
    CLC1SEL2 = 0x15;    /* Configure TMR4_OUT as input for third OR gate */
    CLC1SEL3 = 0x15;    /* Configure TMR4_OUT as input for fourth OR gate */
    /* All four inputs are not inverted*/
    CLC1GLS0 = 0x02;
    CLC1GLS1 = 0x08;
    CLC1GLS2 = 0x20;
    CLC1GLS3 = 0x80;
    CLC1CONbits.EN = 1;      /* CLC1 enabled; */
    CLC1CONbits.MODE = 3;    /* Mode SR latch */
}
```

To measure the internal peripheral signals with the oscilloscope, the following link must be made:

Table 4-2. Peripheral Mapping to I/O Pins for the Example "Using the CLC to Create an LED Dimming Effect"

Internal CIP Signal	Microcontroller Pin
CLC1_OUT	RA2

This is done in the following function:

```
static void PPS_init(void)
{
    RA2PPS = 0x18;    /*Configure RA2 for CLC1 output*/
}
```

View the PIC18F47Q10 Code Example on GitHub
Click to browse repositories

5. References

1. How to install MCC
2. PIC1000: Getting Started with Writing C-Code for PIC16 and PIC18 Technical Brief
3. DS40001725B - MPLAB Code Configurator User's Guide
4. AN2133 - Extending PIC® MCU Capabilities Using CLC
5. TB3133 - Configurable Logic Cell on PIC® Microcontrollers
6. AN2805 - Robust Debouncing with Core Independent Peripherals
7. DS41631B - Configurable Logic Cell Tips 'n Tricks
8. AN2912 - Using CLCs in Real-Time Applications
9. AN1606 - Using the Configurable Logic Cell (CLC) to Interface a PIC16F1509 and WS2811 LED Driver
10. 20007 CIP1 - Applying Configurable Logic Cell CLC to Interconnect Peripheral Functions

6. Revision History

Document Revision	Date	Comments
A	05/2020	Initial document release

The Microchip Website

Microchip provides online support via our website at www.microchip.com/. This website is used to make files and information easily available to customers. Some of the content available includes:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQs), technical support requests, online discussion groups, Microchip design partner program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Product Change Notification Service

Microchip's product change notification service helps keep customers current on Microchip products. Subscribers will receive email notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, go to www.microchip.com/pcn and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Embedded Solutions Engineer (ESE)
- Technical Support

Customers should contact their distributor, representative or ESE for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in this document.

Technical support is available through the website at: www.microchip.com/support

Microchip Devices Code Protection Feature

Legal Notice

Trademarks

ISBN: 978-1-5224-6289-7

Quality Management System

For information regarding Microchip's Quality Management Systems, please visit www.microchip.com/quality.

Worldwide Sales and Service

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
www.microchip.com/support
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Austin, TX
Tel: 512-257-3370

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Novi, MI
Tel: 248-848-4000

Houston, TX
Tel: 281-894-5983

Indianapolis
Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453
Tel: 317-536-2380

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608
Tel: 951-273-7800

Raleigh, NC
Tel: 919-844-7510

New York, NY
Tel: 631-435-6000

San Jose, CA
Tel: 408-735-9110
Tel: 408-436-4270

Canada - Toronto
Tel: 905-695-1980
Fax: 905-695-2078

ASIA/PACIFIC

Australia - Sydney
Tel: 61-2-9868-6733

China - Beijing
Tel: 86-10-8569-7000

China - Chengdu
Tel: 86-28-8665-5511

China - Chongqing
Tel: 86-23-8980-9588

China - Dongguan
Tel: 86-769-8702-9880

China - Guangzhou
Tel: 86-20-8755-8029

China - Hangzhou
Tel: 86-571-8792-8115

China - Hong Kong SAR
Tel: 852-2943-5100

China - Nanjing
Tel: 86-25-8473-2460

China - Qingdao
Tel: 86-532-8502-7355

China - Shanghai
Tel: 86-21-3326-8000

China - Shenyang
Tel: 86-24-2334-2829

China - Shenzhen
Tel: 86-755-8864-2200

China - Suzhou
Tel: 86-186-6233-1526

China - Wuhan
Tel: 86-27-5980-5300

China - Xian
Tel: 86-29-8833-7252

China - Xiamen
Tel: 86-592-2388138

China - Zhuhai
Tel: 86-756-3210040

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-3090-4444

India - New Delhi
Tel: 91-11-4160-8631

India - Pune
Tel: 91-20-4121-0141

Japan - Osaka
Tel: 81-6-6152-7160

Japan - Tokyo
Tel: 81-3-6880- 3770

Korea - Daegu
Tel: 82-53-744-4301

Korea - Seoul
Tel: 82-2-554-7200

Malaysia - Kuala Lumpur
Tel: 60-3-7651-7906

Malaysia - Penang
Tel: 60-4-227-8870

Philippines - Manila
Tel: 63-2-634-9065

Singapore
Tel: 65-6334-8870

Taiwan - Hsin Chu
Tel: 886-3-577-8366

Taiwan - Kaohsiung
Tel: 886-7-213-7830

Taiwan - Taipei
Tel: 886-2-2508-8600

Thailand - Bangkok
Tel: 66-2-694-1351

Vietnam - Ho Chi Minh
Tel: 84-28-5448-2100

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4485-5910
Fax: 45-4485-2829

Finland - Espoo
Tel: 358-9-4520-820

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Garching
Tel: 49-8931-9700

Germany - Haan
Tel: 49-2129-3766400

Germany - Heilbronn
Tel: 49-7131-72400

Germany - Karlsruhe
Tel: 49-721-625370

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Germany - Rosenheim
Tel: 49-8031-354-560

Israel - Ra'anana
Tel: 972-9-744-7705

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Italy - Padova
Tel: 39-049-7625286

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Norway - Trondheim
Tel: 47-72884388

Poland - Warsaw
Tel: 48-22-3325737

Romania - Bucharest
Tel: 40-21-407-87-50

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

Sweden - Gothenberg
Tel: 46-31-704-60-40

Sweden - Stockholm
Tel: 46-8-5090-4654

UK - Wokingham
Tel: 44-118-921-5800
Fax: 44-118-921-5820