Kinetis EA Series Cookbook

Getting started: software examples to exercise microcontroller features

by: Steve Mihalik, Pedro Aguayo, Osvaldo Romero, and Kushal Shah

Contents

1 Introduction

This application note provides software examples and startup code to help users get started with the Kinetis EA series MCUs.

Examples were developed and tested on the FRDMPKEA128 board using S32 Design Studio V1.0. Complete source code and projects are available in a separate zip file at nxp.com.

To access the projects in the zip file, either:

- Windows: unzip archive
- S32 Design Studio: use File - Import and select the unzipped file

Example projects are incorporated in S32 Design Studio V 1.1. These application note projects can be imported into S32DS by selecting File - Import... - S32 Design Studio - S32DS Example Project. Then chose the project of interest.

If new to the KEA Series family, see KEA header file usage for a quick reference on using the header files.

The Kinetis EA series MCU is a highly scalable portfolio of 32-bits ARM Cortex-M0+ MCUs aimed for the automotive markets. The family is optimized for cost-sensitive applications offering low pin-count option with very low power consumption.

2 Software examples

The table below lists the examples in this application note. The three Hello World examples are intended to be base projects. Users can select one and add their own code and/or code from other projects to create a new project.

Table 1. List of Examples

Example	Program Name	Summary
Hello World	hello	Minimal bring up. Exercises GPIO. • An input is polled (Button 0) • An output (to an LED) is set to the opposite state of the input
Hello World + clocks	hello_clocks	Common clock initializations are performed. • Internal Clock Source (ICS) is configured for ◦ FLL engaged internal (FEI) mode (default) ◦ FLL engaged external (FEE) mode • Clock dividers to peripherals are initialized. • Clock frequency can be verified on BUSOUT pin. • A Programmable Interrupt Timer (PIT) is initialized for 1 second. At timeout an output (to LED) is toggled.
Hello World + clocks + interrupts	hello_clocks_interrupts	A simple interrupt is configured and serviced. • A Programmable Interrupt Timer (PIT) is initialized for a 1 second timeout • At PIT's timeout, an interrupt is taken • The interrupt handler toggles an output (to an LED)
Timed I/O	FTM_PWT	Common timed I/O functions are performed: FTM (FlexTimer Module): • Initialization of module's counter • Pulse Width Modulation • Output Compare • Input capture PWT (Pulse Width Timer): • Measures time clocks between two edges
Analog-to-digital converter (ADC)	ADC	A basic analog to digital conversion is performed: • ADC configured for SW trigger, continuous mode • ADC channel 10 (connected to pot) is converted • Result is scaled to 0 – 5000 mV • LEDs are lit on evaluation board pre result • ADC channel 10 (connected to pot) is converted • Result is scaled to 0 – 5000 mV
UART	UART	Transmits and receives characters • UART is initialized for 9600 baud, 1 stop, no parity • A string is transmitted, then a prompt character • A character is received then echoed back

2.1 Hello World

2.1.1 Description

Summary: This short project is a starting point with minimal code that only exercises a GPIO input and output. A GPIO input is polled to detect a high or low on the pin. A GPIO output is set depending on the input state. If running this code on the Freedom plus KEA128 evaluation board, pressing button 0 lights up the blue LED per the diagram below.

See KEA header file usage for a summary of how header files are used.

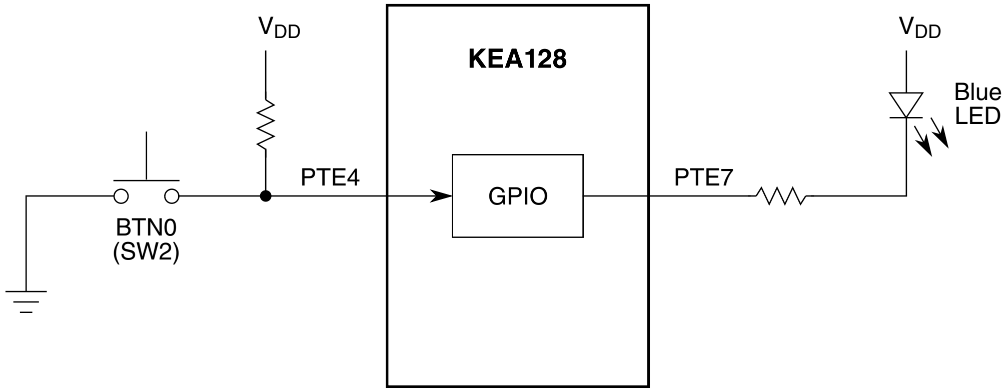

Figure 1. Hello World example block diagram

Registers used for GPIO configuration are summarized below.

Table 2. GPIO configuration registers summary

Operation	Module Register	Register Name	Comment
Port assignment	SIM_PINSELx	Pin Select	Register is not used for KEA GPIO. Used for peripheral I/O.
I/O direction	GPIOx_PDDR	Port Data Direction Register	0: Input (default) 1: Output
Input Disable	GPIOx_PIDR [PID],	Port Input Disable Register	0: Input enabled 1: Input disabled (default)
Internal pullup/down	PORT_PUE0x [PTxPEn]	Pull Up Enable	0: Input enabled (default) 1: Input disabled
Input glitch filter	PORT_IOFLTx [FLTDIVn]	Port Filter Register	0: Input enabled (default) 1: Input disabled
High current drive	PORT_HDRVE [PTxn]	Port High Driver Enable Register	0: Input enabled (default) 1: Input disabled

Registers used for GPIO read and write of I/O are summarized below.

Table 3. GPIO read and write of I/0 registers

Operation	Module Register	Register Name	Comment
Pad data in	GPIOx_PDIR[n]	Port Data Input Register	Used to read input pin state
Pad data out	GPIOx_PDOR[n]	Port Data Output Register	Controls pin output; allows reading pin state.
Pad data out (with masking)	GPIOx_PSOR[n] GPIOx_PCOR[n] GPIOx_PTOR[n]	Port Set Output Register Port Clear Output Register Port Toggle Output Register	Selected port pins are set, cleared or toggled without affecting other pins

Each 8-bit port pin is mapped to 32-bit GPIO/FGPIO registers as shown below.

Register bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Port pin	PTD7	PTD6	PTD5	PTD4	PTD3	PTD2	PTD1	PTD0	PTC7	PTC6	PTC5	PTC4	PTC3	PTC2	PTC1	PTC0	PTB7	PTB6	PTB5	PTB4	PTB3	PTB2	PTB1	PTB0	PTA7	PTA6	PTA5	PTA4	PTA3	PTA2	PTA1	PTA0

Figure 2. KEA128 GPIOA/FGPIOA register bits assignment

Register bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Port pin	PTH7	PTH6	PTH5	PTH4	PTH3	PTH2	PTH1	PTH0	PTG7	PTG6	PTG5	PTG4	PTG3	PTG2	PTG1	PTG0	PTF7	PTF6	PTF5	PTF4	PTF3	PTF2	PTF1	PTF0	PTE7	PTE6	PTE5	PTE4	PTE3	PTE2	PTE1	PTE0

Figure 3. KEA128 GPIOB/FGPIOB register bits assignment

Register bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Port pin	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	PTI6	PTI5	PTI4	PTI3	PTI2	PTI1	PTI0

Figure 4. KEA128 GPIOC/FGPIOC register bits assignment

2.1.2 Design steps

- Perform initializations before main:
  - Initialize stack pointer
  - Write initial values to initialized variables
  - Other normal compiler initializations
- Configure a port pin as output (goes to LED on evaluation board)
- Configure a port pin as input (from push button 0 on evaluation board)
- Loop forever:
  - If pushbutton is not pressed (PTE7 input = 1)
    - Turn LED off (output = 1)
  - Else (input = 0)
    - Turn LED on (output = 0)

2.1.3 Code

2.1.3.1 hello.c

```
#include "derivative.h" /* include peripheral declarations SKEAZ128M4 */
#define PTE7 7          /* Port PTE7, bit 7: output to blue LED */
#define PTE4 4          /* Port PTE4, bit 4: input from BTN0*/

void main(void) {
                         /* Configure port E4 as GPIO input (BTN 0 [SW2] on EVB) */
  GPIOB_PDDR &= ~(1<<PTE4); /* Port E4: Data Direction= input (default) */
  GPIOB_PIDR &= ~(1<<PTE4); /* Port E4: Input Disable= 0 (input enabled) */
  PORT_PUE0 |= 0<<PTE4;     /* Port E4: No internal pullup (default) */

                            /* Configure port E7 as GPIO output (LED on EVB) */
  GPIOB_PDDR |= 1<<PTE7;    /* Port E7: Data Direction= output */
  GPIOB_PIDR &= 1<<PTE7;    /* Port E7: Input Disable= 1 (default) */

  for(;;) {
    if (GPIOB_PDIR & (1<<PTE4))  {  /* If Pad Data Input = 1 (BTN0 [SW2] not pushed) */
      GPIOB_PSOR |= 1<<PTE7;        /* Set Output on port E7 (LED off) */
    }
    else {                         /* If BTN0 was pushed */
      GPIOB_PCOR |= 1<<PTE7;       /* Clear Output on port E7 (LED on) */
    }
  }
}
```

2.2 Hello World + clocks

2.2.1 Description

Summary: This project primarily expands the prior hello world project by adding common clock operation modes:

- FEI (FLL Engaged Internal)
- FEE (FLL Engaged External)

The functions that configure the clock modes are also used in other examples but included as separate files clocks.c and clocks.h.

NOTE

Other KEA Series family members have clock differences.

To allow observing clock frequency two options are initialized:

- FEI mode:
  - Insert a breakpoint in the code after the FEI initialization function.
  - At the breakpoint, measure prescaled bus clock at PTH2 with an oscilloscope.
  - The frequency on PTH2 will be: 24 MHz Bus Clock / 128 = ~187.5 kHz.
- FEE mode: Either
  - Measured PTH2 as in FEI mode. The frequency will be: 20 MHz Bus Clock / 128 = 156.25 kHz.
  - Measure or observe the blue LED on the evaluation board toggling every: 20 M PIT clocks x 1 sec/20 M clocks = 1 second

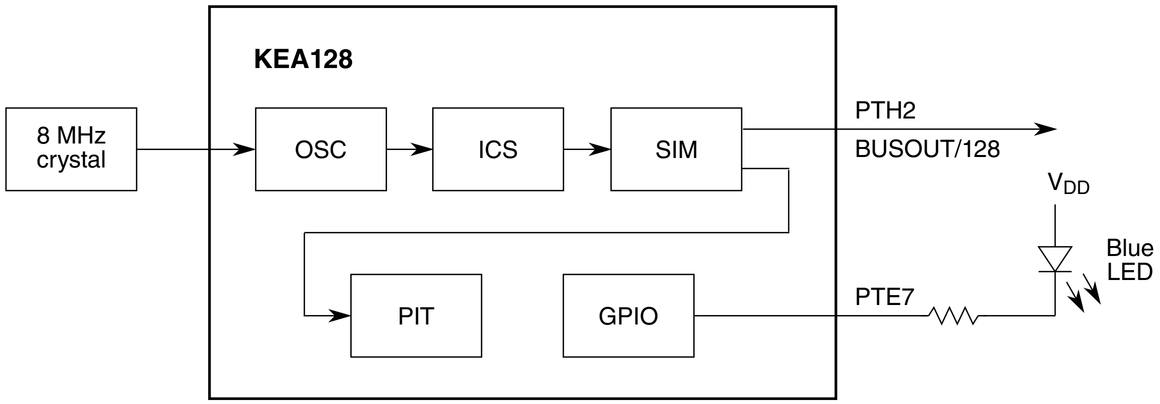

Figure 5. Block diagram of Hello World and Clocks

The overall clock system is shown in the figure below from the KEA128 Reference Manual.

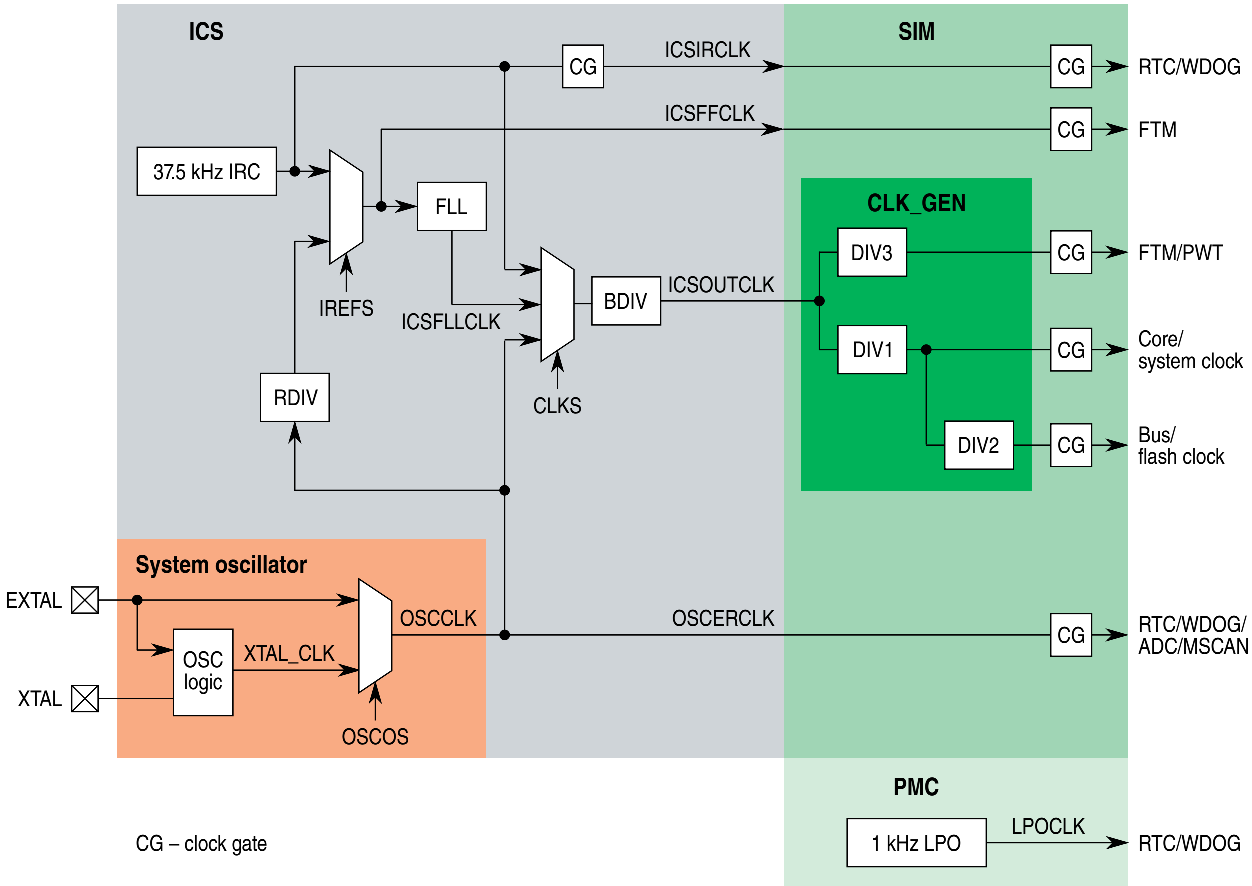

Figure 6. Overall clock system block diagram

The FEI clock mode starts with the internal reference clock with a frequency of 37.5 kHz on KEA128. This FEI example's internal clock flow is shown below.

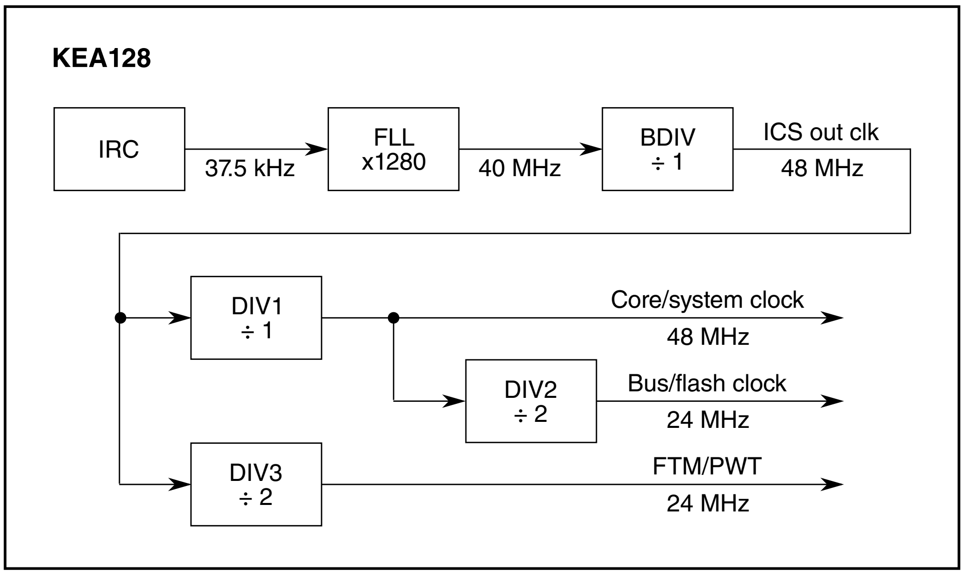

Figure 7. FEI example internal clock flow

The FEE clock mode starts with the external crystal which as a frequency of 8 MHz. This FEE example's internal clock flow is shown below.

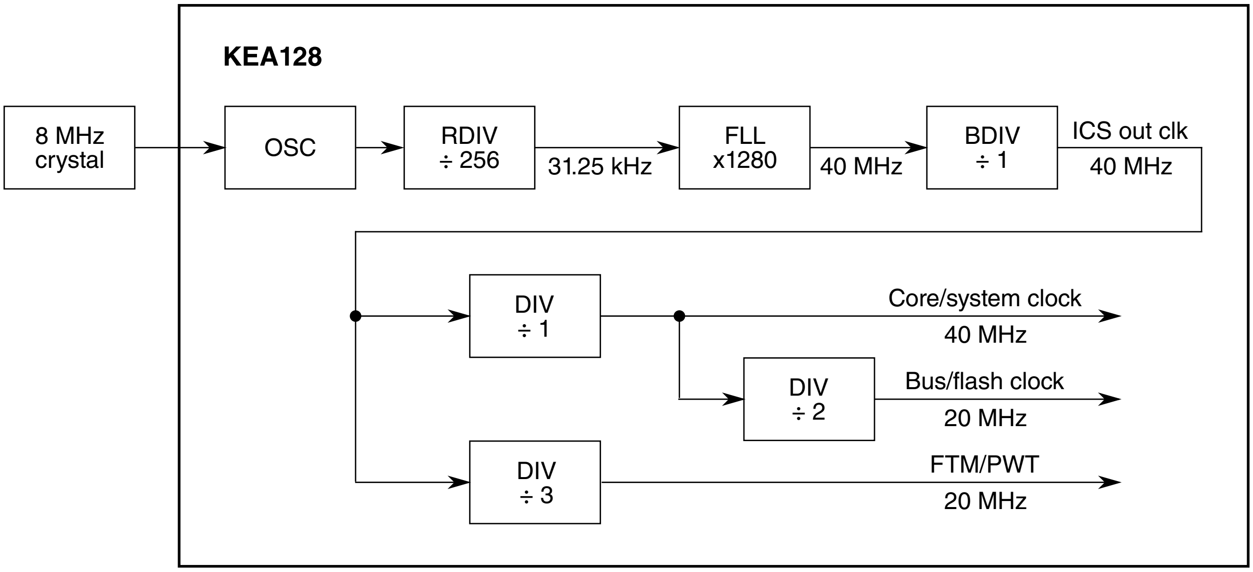

Figure 8. FEE example internal flow

2.2.2 Design steps

- Initialize GPIO output on PTE7 to LED for use of displaying PIT timeouts
- Enable BUSOUT on PTH2
  - Allows verifying expected bus frequency after clock initializations
- Initialize clocks in FLL Engaged Internal mode, 48 MHz core & 24 MHz bus clocks
- Initialize clocks in FLL Engaged External mode, 40 MHz core & 20 MHz bus clocks
- Initialize PIT to count 20 million clocks
- On PIT timeout, toggle output to LED (approximately 1 s).

FEE mode clock BUSOUT (BUSCLOCK/128) output at 156.25 kHz frequency is shown below.

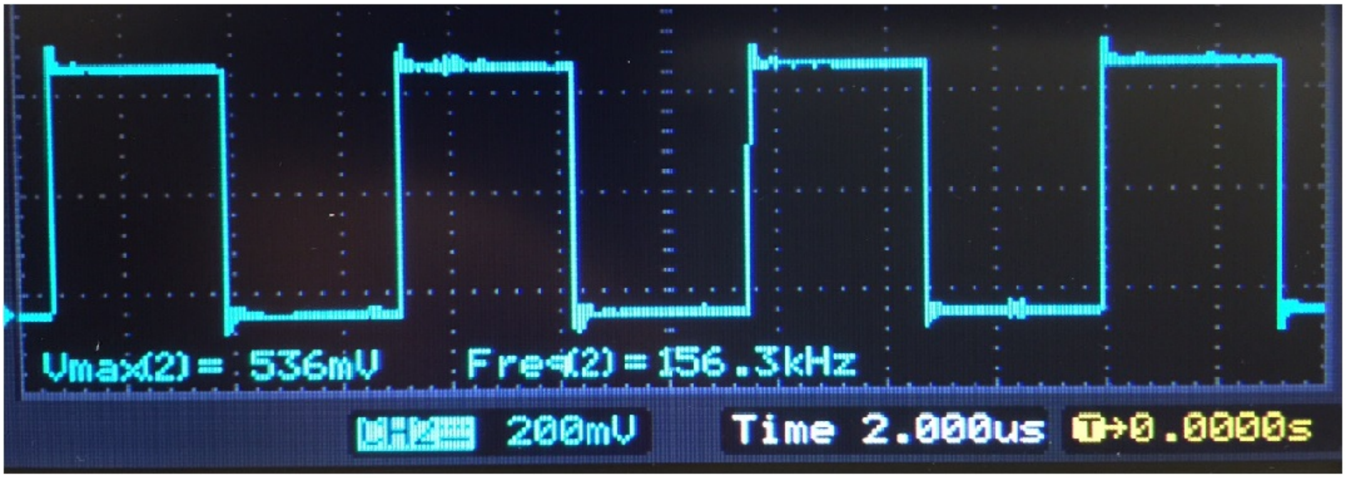

Figure 9. FEE mode clock BUSOUT (BUSOUT/128) output at 156.25 KHz frequency scope shot

2.2.3 Code

2.2.3.1 hello_clocks.c

```
#include "derivative.h"          /* include peripheral declarations S32K144 */
#define PTE7 7                   /* Port PT7, bit 7: output to blue LED */

int pit0_flag_counter = 0;          /* Counter for PIT0 timer expirations */

void init_clks_FEI_48MHz (void) {  /* FLL Enabled with Internal clock */
  OSC_CR = 0x00;         /* (default value) */
                         /* OSCEN=0: OSC module disabled */
                         /* OSCSTEN=0: OSC clock disabled in Stop mode */
                         /* OSCOS=0: Ext clk source (don't care here */
                         /* RANGE=0: Low Freq range of 32 KHz */
                         /* HGO=0: low power High Gain Osc mode (don't care here */
  ICS_C2 = 0x20;         /* Use defaults until dividers configured (default) */
                         /* BDIV=1:  divided by 2  */
                         /* LP = 0:  FLL Not disabled in bypass mode  */
  ICS_C1 = 0x04;         /* Internal ref clock is FLL source (default)*/
                         /* CLKS=0: Output of FLL is selected to control bus freq */
                         /* RDIV=0: Ref divider = 1 since RANGE = 0 */
                         /* IREFS=0: Int Ref clock is selected */
                         /* IRCLKEN=0: ICSIRCLK is inactive */
                         /* IREFSTEN=0: Int ref clk is disabled in Stop mode */
  while ((ICS_S & ICS_S_LOCK_MASK) == 0); /* Wait for FLL to lock*/
  SIM_CLKDIV = 0x01100000; /* OUTDIV1 = 0; Core/sysclk is ICSOUTCLK div by 1 */
                             /* OUTDIV2 = 1  bus/flash is OUTDIV1/2 */
                             /* OUTDIV3 = 1; FTMs, PWT is ICSOUTCLK div by 2 */
  ICS_C2 = 0x00;             /* BDIV div by 1- increases bus/flash freq */
 }

void init_clks_FEE_40MHz(void) {    /* FLL Enabled with External clock */
  OSC_CR = 0x96;         /* High range & gain; select osc */
                         /* OSCEN =1 ; OSC module enabled */
                         /* OSCSTEN = 0; OSC clock disabled in stop mode */
                         /* OSCOS = 1; OSC clock source is selected */
                         /* RANGE = 1; High freq range of 4-24 MHz */
                         /* HGO = 1; High-gain mode */
  while ((OSC_CR & OSC_CR_OSCINIT_MASK) == 0); /* Wait until oscillator is ready*/
  ICS_C2 = 0x20;        /* BDIV div by 2; use default until dividers configured*/
                        /* LP = 0; FLL is not disabled in bypass mode */
  ICS_C1 = 0x18;        /* 8 Mhz ext ref clk/256 is source to FLL */
                        /* CLKS = 0; Output of FLL is selected (default) */
                        /* RDIV = 3; ref clk prescaled by 256 with RANGE=0 */
                        /* IREFS = 0; ext clk source selected */
                        /* IRCLKEN = 0; ICSIRCLK inactive */
                        /* IREFSTEN = 0; Int ref clk disabled in Stop mode */
  while ((ICS_S & ICS_S_IREFST_MASK) == 1);    /* Wait for external source selected */
  while ((ICS_S & ICS_S_LOCK_MASK) == 0);      /* Wait for FLL to lock */
  SIM_CLKDIV = 0x01100000; /* OUTDIV1 = 0; Core/sysclk is ICSOUTCLK div by 1 */
                             /* OUTDIV2 = 1  bus/flash is OUTDIV1/2 */
                             /* OUTDIV3 = 1; FTMs, PWT is ICSOUTCLK div by 2 */
  ICS_C2 = 0x00;             /* BDIV div by 1- increases bus/flash freq */
 }

void init_PIT(void) {
  SIM_SCGC |= SIM_SCGC_PIT_MASK;      /* Enable bus clock to PIT module */
  PIT_MCR = 0x0;                      /* Turn on PIT module, Freeze disabled */
  PIT_LDVAL0 = 20000000 - 1;          /* PIT0: Load value to count 20M bus clocks */
  PIT_TCTRL0 |= PIT_TCTRL_TEN_MASK;  /* PIT0: Start timer */
}

int main(void) {

  GPIOB_PDDR |= 1<<PTE7; /* Port Data Dir: enable output on port E7 (blue LED) */
  SIM_SOPT0|=SIM_SOPT0_CLKOE_MASK|SIM_SOPT0_BUSREF(0b111); /*Enable BUSOUT on PTH2 */
```

```
  init_clks_FEI_48MHz();  /* KEA128 clks FEI (default): core ~48 MHz, bus ~24MHz */
  init_clks_FEE_40MHz();  /* KEA128 clks FEE: core 40 MHz, bus 20MHz */

  init_PIT();               /* Initialize PIT 0 for 1 second timeout @ 20MHz bus clock */
  for (;;) {                /* Toggle output to LED every 20M bus clocks */
    while (0 == (PIT_TFLG0 & PIT_TFLG_TIF_MASK)) {}   /* Wait for PIT0 flag */
    pit0_flag_counter++;              /* PIT0 expired. Increment counter */
    GPIOB_PTOR |= 1<<PTE7;            /* Toggle Output on port E7 (blue LED) */
    PIT_TFLG0 |= PIT_TFLG_TIF_MASK;   /* Clear PIT0 flag */
 }
}
```

2.3 Hello World + Interrupts

2.3.1 Description

Summary: This project expands the prior hello_clocks project by replacing polling for the PIT flag with using the PIT channel's interrupt.

Interrupt source vectors and numbers are found in the reference manual for the device. From the partial interrupt vector assignments table of the KEA128 reference manual (below), PIT Channel 0 has:

- vector 38
- IRQ 22
- Uses NVIC Interrupt Priority Register

Table 4. Interrupt vector assignment

Vector	IRQ	NVIC IPR register number	Source module
38	22	5	PIT_CH0

2.3.2 Design steps

- Initialize a pin for output (to LED on Freedom + evaluation board)
- Initialize clocks for FEE mode with 40 MHz core clock
- Initialize interrupts (only PIT channel 0 here):
    - Clear any prior pending PIT channel 0 interrupt
    - Enable PIT channel 0 interrupt
    - Set PIT channel 0 interrupt priority from 0 to 3 (3 is highest)
- Initialize PIT
    - Enable bus clock
    - Turn on PIT
    - Load PIT channel timer value of clocks to count
    - Enable PIT channel interrupt
    - Start PIT channel timer
- Wait forever: Interrupt code
- PIT channel 0 ISR:
    - Increment counter
    - Toggle output
    - Clear PIT channel flag

2.3.3 Code

2.3.3.1 hello_interrupts.c

```
#include "derivative.h"         /* include peripheral declarations SKEAZ128M4 */
#include "clocks.h"

#define PTE7 7                  /* Port PT7, bit 7: output to blue LED */

int pit0_flag_counter = 0;      /* Counter for PIT0 timer expirations */

void init_IRQs (void) {
  NVIC_ClearPendingIRQ(PIT_CH0_IRQn);  /* Clear any Pending IRQ for all PIT ch0 (#22) */
  NVIC_EnableIRQ(PIT_CH0_IRQn);        /* Set Enable IRQ for PIT_CH0 */
  NVIC_SetPriority(PIT_CH0_IRQn,0);    /* Set Priority for PIT_CH0 */
}

void init_PIT(void) {
  SIM_SCGC |= SIM_SCGC_PIT_MASK;     /* Enable bus clock to PIT module */
  PIT_MCR = 0x0;                     /* Turn on PIT module, Freeze disabled */
  PIT_LDVAL0 = 20000000 - 1;         /* PIT0: Load value to count 20M bus clocks */
  PIT_TCTRL0 |= PIT_TCTRL_TIE_MASK;  /* Enable interrupt */
  PIT_TCTRL0 |= PIT_TCTRL_TEN_MASK;  /* Enable (start) timer */
}

int main(void) {
  int idle_counter = 0;
  GPIOB_PDDR |= 1<<PTE7;    /* Port Data Dir: output on port E7, blue LED */
  init_clks_FEE_40MHz();    /* Initialize FLL: 8MHz xtal, 40 MHz core, 20 MHz bus clks */
  init_IRQs();              /* Initialize interrupts: enable, priorities */
  init_PIT();               /* Initialize PIT0: 1 sec timeout, IRQ enabled */
  for(;;) {
    idle_counter++;
  }
}

void PIT_CH0_IRQHandler (void) {
  pit0_flag_counter++;            /* PIT0 expired. Increment counter */
  GPIOB_PTOR |= 1<<PTE7;          /* Toggle Output (1) on port E7 (blue LED) */
  PIT_TFLG0 |= PIT_TFLG_TIF_MASK; /* Clear PIT0 flag */
}
```

2.3.3.2 clocks.c

```
void init_clks_FEE_40MHz(void) {     /* FLL Enabled with External clock */
  OSC_CR = 0x96;       /* High range & gain; select osc */
                       /* OSCEN =1 ; OSC module enabled */
                       /* OSCSTEN = 0; OSC clock disabled in stop mode */
                       /* OSCOS = 1; OSC clcok source is selected */
                       /* RANGE = 1; High freq range of 4-24 MHz */
                       /* HGO = 1; High-gain mode */
  while ((OSC_CR & OSC_CR_OSCINIT_MASK) == 0); /* Wait until oscillator is ready*/
  ICS_C2 = 0x20;      /* BDIV div by 2; use default until dividers configured*/
                      /* LP = 0; FLL is not disabled in bypass mode */
  ICS_C1 = 0x18;      /* 8 Mhz ext ref clk/256 is source to FLL */
                      /* CLKS = 0; Output of FLL is selected (default) */
                      /* RDIV = 3; ref clk prescaled by 256 with RANGE=0 */
                      /* IREFS = 0; ext clk source selected */
                      /* IRCLKEN = 0; ICSIRCLK inactive */
                      /* IREFSTEN = 0; Int ref clk disabled in Stop mode */
  while ((ICS_S & ICS_S_IREFST_MASK) == 1);    /* Wait for external source selected */
```

```
  while ((ICS_S & ICS_S_LOCK_MASK) == 0);       /* Wait for FLL to lock */
  SIM_CLKDIV = 0x01100000; /* OUTDIV1 = 0; Core/sysclk is ICSOUTCLK div by 1 */
                           /* OUTDIV2 = 1  bus/flash is OUTDIV1/2 */
                           /* OUTDIV3 = 1; FTMs, PWT is ICSOUTCLK div by 2 */
  ICS_C2 = 0x00;           /* BDIV div by 1- increases bus/flash freq */
}
```

2.4 Timed I/O (FTM, PWT)

2.4.1 Description

Summary: Common digital I/O functions are performed using the FlexTimer Module (FTM) and Pulse Width Timer (PWT). The FTM modules are used to implement Pulse Width Modulation (PWM), Output Compare (OC) and Input Capture (IC) mode examples. Each FTM uses a single counter as a time base for its channels. The PWT measures time between edges, either rising, falling or either edge, to measure pulse width in terms of its clock counts.

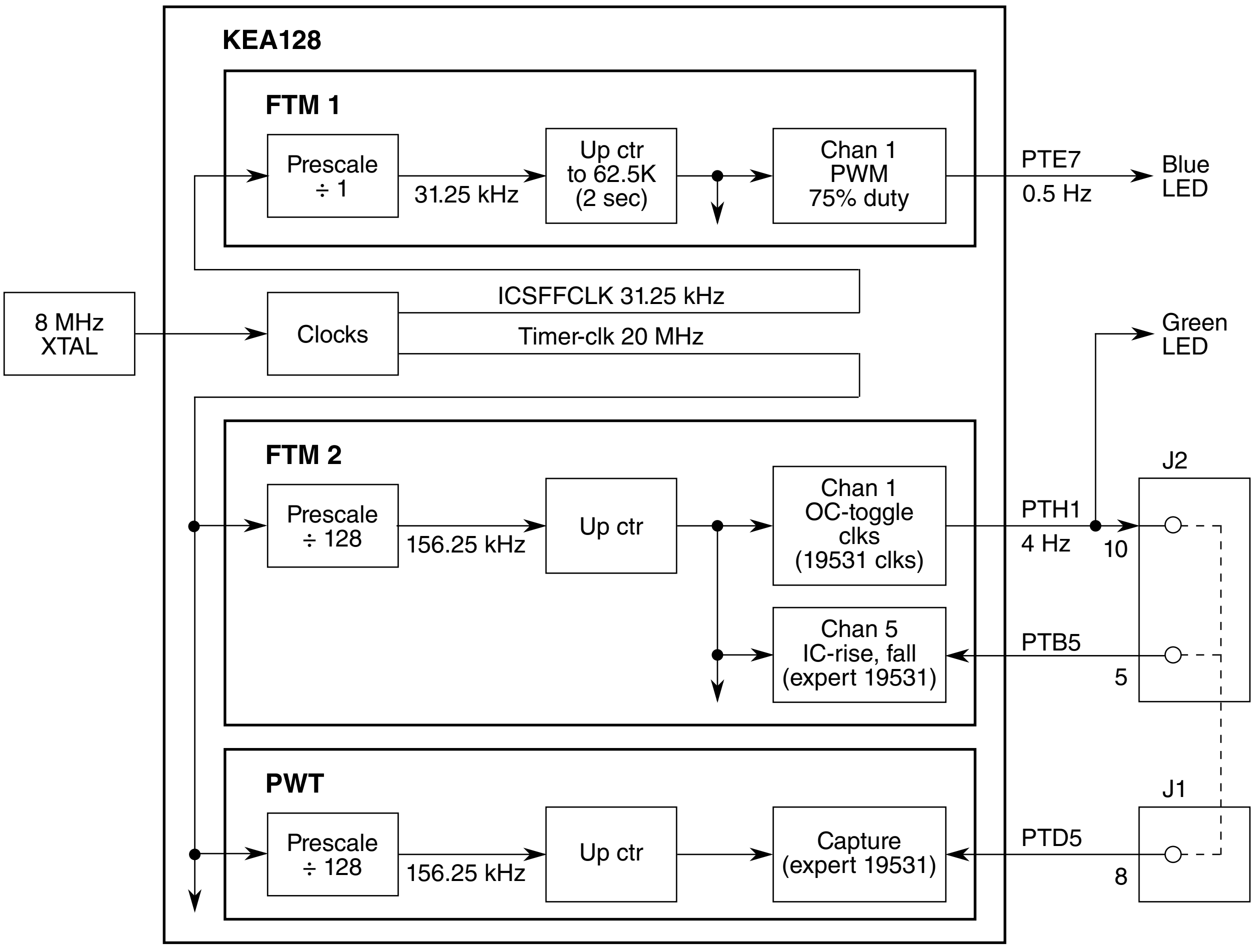

Figure 10. Timed I/O example block diagram

The above diagram shows the configurations used for the channels. Each FlexTimer module and PWT module have multiple clock sources that can be selected. For this example the TIMER_CLK, configured for 20 MHz, is the clock source for FTM2 and PWT. The ICSFFCLK, configured for 31.25KHz, is the FTM1 clock source. The clock initialization is the same as the FEE mode configuration in the hello_clocks project. The following table summarizes each channels configuration to achieve the desired timing.

Table 5. Timed I/O example channel configurations

Timed I/O Function	Channel	Module's Clock Input[1]	Prescaled Module Clock	Channel Configuration
PWM	FTM1 Chan 1	31.25 kHz IRCFFCLK	31.25 kHz	Up counts to 62.5 K clocks (2 s) with 75% duty cycle.
Output Compare	FTM2 Chan 1	20 MHz TIMER_CLK	156.25 kHz	Toggle output every 19531 clocks (125 ms).
Input capture	FTM2 Chan 5			Captures counter value on rising or falling edge. Expected delta from two edges = 19531.
Pulse Width Timer	PWT	20 MHz TIMER_CLK	156.25 kHz	Captures counter value at two edges. Configured to capture count between rising and falling edges. Expected count = 19531

1. See Hello World + clocks example, FEE clock mode

See result waveforms below. The PWM on the bottom has a 2 second period, and the upper Output Compare function toggles every 0.125 seconds.

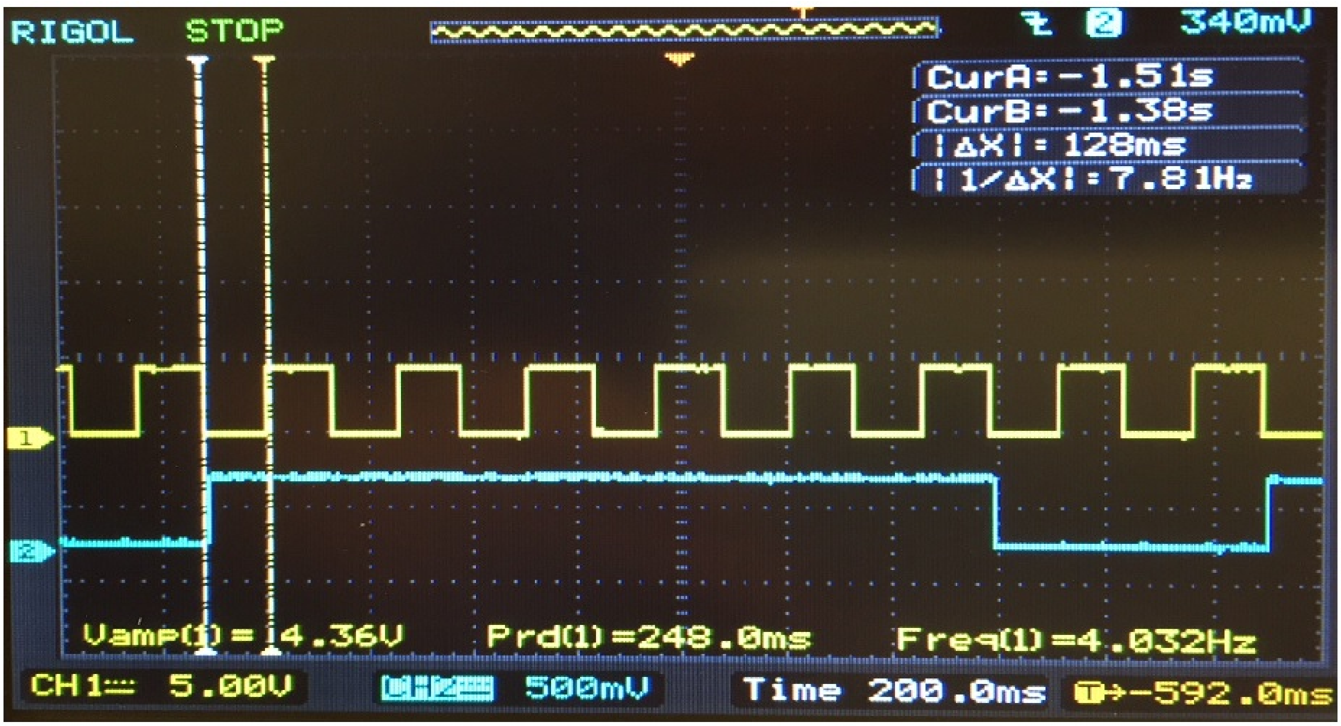

Figure 11. Timed I/O example waveforms

Clock sources to FTM and PWT modules have a range of selection options as shown below. This example uses ICSFLL clock for a slow clock, and only to FTM1. FTM2 and PWT use the faster TIMER_CLK so their functions are synchronized.

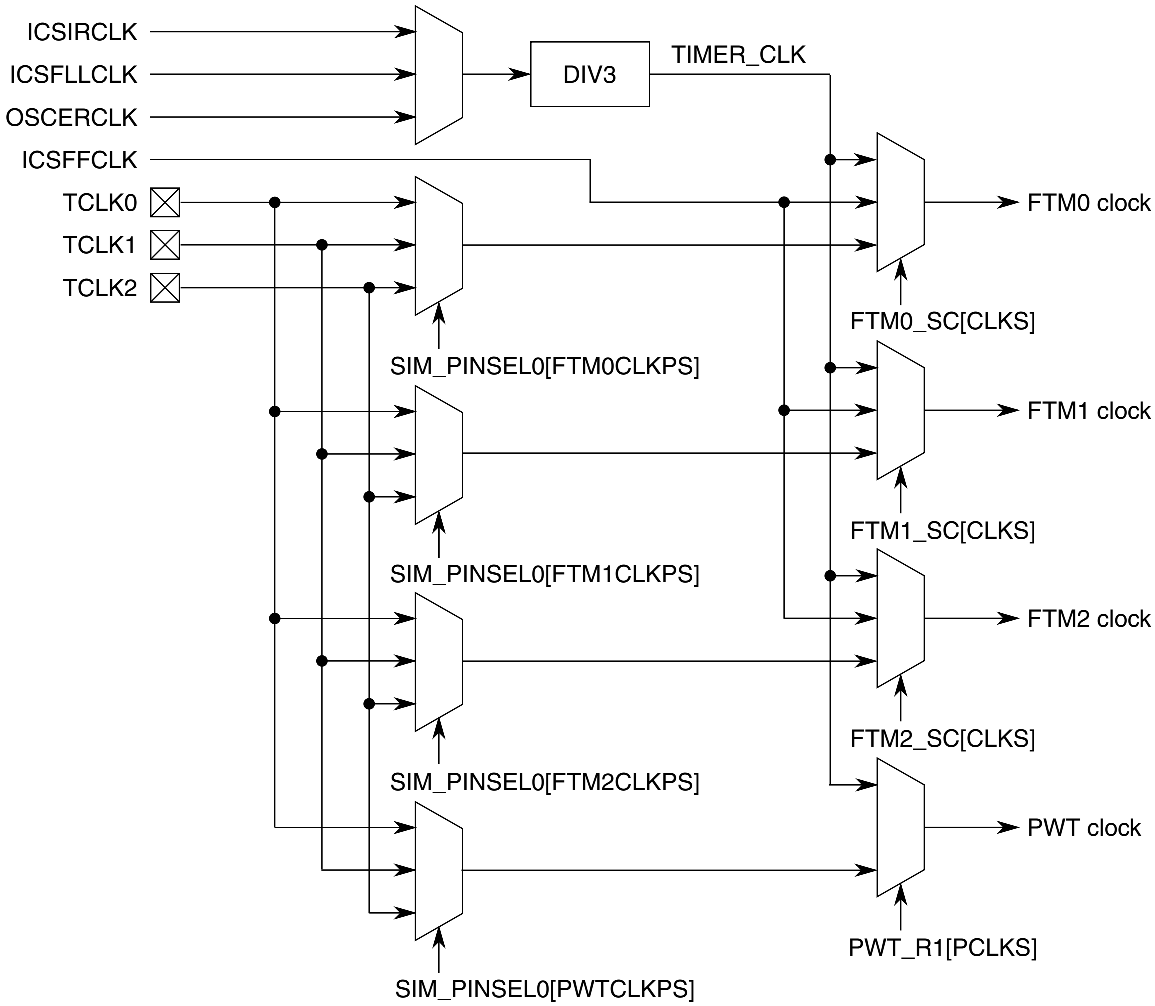

Figure 12. Timed I/O example clock sources

Channel modes are configured by settings in registers for the entire FTM module and individual channel. The following table shows the settings used for the channel functions implemented in this example.

Not all FTM modules implement all features or registers. See the Chip Configuration chapter in the reference manual to see what is included. For example, on KEA128 FTM0 and FTM1 do not implement the COMBINE register, so those modules do not include FTM modes that combine pairs of channels. For a complete list of FTM channel modes and their required settings see the FTM chapter in the reference manual.

Table 6. Timed I/O example required settings for FTM channel modes

Module Register [Bit field] Settings			**Channel Register [Bit field] Settings**		**Mode**	**Configuration**
COMBINE[1] [DECAPEN]	COMBINE[1] [COMBINE]	SC [CPWMS]	C*n*SC [MSnB:MSnA]	C*n*SC [ELSnB:ELSnA]		
0	0	0	00	11	Input Capture	Capture on rising or falling edge
0	0	0	01	01	Output Compare	Toggle output on match
0	0	0	1X	X1	Edge-Aligned PWM	Low-True pulses (set Output on match)

1. COMBINE register is not implemented on some FlexTimer modules. In this case, DECAPEN and COMBINE bit fields are not applicable

2.4.2 Design steps

- Initialize clocks to FEE mode, 40 MHz system clock, 20 MHz TIMER_CLK
- Initialize FTM modules registers
- Initialize FTM 1 channel 1 to Pulse Width Modulation (PWM) mode
- Initialize FTM 2 channel 1 to Output Compare (OC) mode
- Initialize FTM 2 channel 5 to Input Capture (IC) mode
- Initialize PWM
- Loop forever:
  - If Output Compare flag is set,
    - Toggle pin
    - Clear flag
    - Reload timeout value
  - If Input Capture flag is set,
    - Clear flag
    - Calculate number of clocks since last edge
  - If Pulse Width Timer flag is set,
    - Clear flag
    - Read pulse width

2.4.3 Code

2.4.3.1 main.c

```
#include "derivative.h" /* include peripheral declarations SKEAZ128M4 */
#include "FTM.h"
#include "PWT.h"
#include "clocks.h"

void init_clk_FEE_40MHz(void);

int main(void) {

  init_clks_FEE_40MHz();        /* KEA128 clks FEE, 8MHz xtal: core 40 MHz, bus 20MHz */
  init_FTM ();                  /* Enable bus clock to FTM1,2 prescaled by 128 */
  init_FTM1_ch1_PWM();          /* PTE7 output, to blue LED */
  init_FTM2_ch1_OC();           /* PTH1 output, to green LED & J2_10 */
  init_FTM2_ch5_IC();           /* PTB5 input; connect J2_5 and J2_10  */
  init_PWT();                   /* PTD5 input */
  start_FTM_counters();

  for(;;) {                    /* Poll to look for timed I/O flags */
    output_compare_FTM2_ch1(); /* If output compare match: */
                               /*   then toggle pin, clear flag, reload timer */
    input_capture_FTM2_ch5();  /* If input captured: clear flag, read timer */
    pulse_width_timer_PWT();   /* If two falling edges captured: */
                               /*   then clear flag, read pulse width value */
    }
}
```

2.4.3.2 FTM.c

```
#include "derivative.h" /* include peripheral declarations SKEAZ128M4 */
#include "FTM.h"

uint16_t CurrentCaptureVal = 0;
uint16_t PriorCaptureVal = 0;
uint16_t DeltaCapture = 0;

void init_FTM(void) {
  SIM_SCGC |= SIM_SCGC_FTM1_MASK /* Sys Clk Gate Ctrl: enable bus clock to FTM1,2 */
           | SIM_SCGC_FTM2_MASK;

                         /* FTM1 module settings for desired channel modes: */
  FTM1_SC = 0x00000000;
                         /* CWMS (Center aligned PWM Select) = 0 (default, up count) */
                         /* TOIE (Timer Overflow Interrupt Ena) = 0 (default) */
                         /* CLKS (Clock source) = 0 (default, no clock; FTM disabled) */
                         /* PS (Prescaler factor) = 0 (default). Prescaler = 2**0 = 1 */
  FTM1_MOD = 62500 ;     /* FTM1 counter final value (used for PWM mode) */
                         /* FTM1 Period = MOD-CNTIN+0x0001 ~= 62.5K ctr clks  */
                         /*      62.5K clks x 1 sec/31.25K clks = 2 sec (0.5 Hz)  */

                         /* FTM2 module settings for desired channel modes: */
  FTM2_MODE |= FTM_MODE_WPDIS_MASK; /* Write protect to registers disabled (default) */
  FTM2_COMBINE = 0x0;    /* DECAPEN (Dual Edge Capture Mode Enable) = 0 (default) */
                         /* COMBINE (chans n & n+1) = 0 (default; independent chans) */
  FTM2_SC = 0x00000007;  /* CWMS (Center aligned PWM Select) = 0 (default, up count) */
                         /* TOIE (Timer Overflow Interrupt Ena) = 0 (default) */
                         /* CLKS (Clock source) = 0 (default, no clock; FTM disabled) */
                         /* PS (Prescaler factor) = 7. Prescaler = 2**7 = 128 */
}

void init_FTM1_ch1_PWM(void) {
  FTM1_C1SC = 0x00000028;         /* FTM1 ch1: edge-aligned PWM, low true pulses */
                                  /* CHIE (Chan Interrupt Ena) = 0 (default) */
                                  /* MSB:MSA (chan Mode Select) = 0b10 */
                                  /* ELSB:ELSA (chan Edge or Level Select) = 0b10 */
  FTM1_C1V =  46875;              /* FTM1 ch1 compare value (~75% duty cycle) */
  SIM_PINSEL0 |=SIM_PINSEL_FTM1PS1_MASK; /* Pin Selection: FTM1 CH1 on PTE7 (blue LED) */
}

void init_FTM2_ch1_OC(void) {
  FTM2_C1SC = 0x00000014;        /* FTM2 ch1: Output Compare, toggle output on match */
                                 /* CHIE (Chan Interrupt Ena) = 0 (default) */
                                 /* MSB:MSA (chan Mode Select) = 0b01 */
                                 /* ELSB:ELSA (chan Edge or Level Select) = 0b01 */
  FTM2_C1V = 19531;              /* FTM2 ch 1 Compare Value = 19531  clks */
                                 /* 19531 clks x 1 sec/156.25K clks = 125 msec toggle */
  FTM2_POL &= ~FTM_POL_POL1_MASK; /* Chan 1 polarity = 0 (Default, active high) */
  SIM_PINSEL1 |=SIM_PINSEL1_FTM2PS1(1); /* Map FTM2 CH1 to pin PTH1  (green LED) */
}

void init_FTM2_ch5_IC(void) {
  FTM2_C5SC = 0x0000000C;       /* FTM2 ch5: Input Capture on rising or falling edge */
                                /* CHIE (Chan Interrupt Ena) = 0 (default) */
                                /* MSB:MSA (chan Mode Select) = 0b00 */
                                /* ELSB:ELSA (chan Edge or Level Select) = 0b11 */
  SIM_PINSEL1 &= ~SIM_PINSEL1_FTM2PS5_MASK; /* Use default pad PTB5 */
}

void start_FTM_counters (void) {
  FTM1_SC |= FTM_SC_CLKS(2);  /* Start FTM1 ctr with clk source ICSFFCLK (31.25 KHz) */
  FTM2_SC |= FTM_SC_CLKS(1);  /* Start FTM2 ctr with clk source TIMER_CLK (20 MHz)*/
}

void output_compare_FTM2_ch1() {
  if (1==((FTM2_C1SC & FTM_CnSC_CHF_MASK)>>FTM_CnSC_CHF_SHIFT)) { /* If chan flag is set */
   FTM2_C1SC &= ~FTM_CnSC_CHF_MASK;  /* Clear chan flag: read reg then write 0 to CHF bit*/
```

```
    FTM2_C1V = FTM2_C1V + 19531 ;      /* Update compare value: add 19531 to current value*/
  }
}

void input_capture_FTM2_ch5(void) {
  if (1==((FTM2_C5SC & FTM_CnSC_CHF_MASK)>>FTM_CnSC_CHF_SHIFT)) { /* If chan flag is set */
    FTM2_C5SC &= ~FTM_CnSC_CHF_MASK;  /* Clear chan flag: read reg then write 0 to CHF */
    PriorCaptureVal = CurrentCaptureVal; /* Record value of prior capture */
    CurrentCaptureVal = FTM2_C5V;        /* Record value of current capture */
    DeltaCapture = CurrentCaptureVal - PriorCaptureVal;
           /* Delta Capture will be 19531 if connected to FTM2_ch1 */
  }
}
```

2.4.3.3 PWT.c

```
#include "derivative.h" /* include peripheral declarations SKEAZ128M4 */
#include "PWT.h"

uint16_t PulseWidth = 0;

void init_PWT(void)  {
  SIM_SCGC |= SIM_SCGC_PWT_MASK; /* Enable Clock to PWT */
  PWT_R1 = 0x00001780;  /* Initialize PWT for measuring falling edges */
                        /* PCLKS (PWT Clock Source Select) = 0 (default, BUS_CLK) */
                        /* PINSEL (PWT Pulse Input Selection) = 0 (default, PWTIN[0]) */
                        /* EDGE (PWT Input Edge sensitivity) = 2: */
                               /* 1st falling edge starts PW measurement. */
                               /* PW captured on all subsequent falling edges. */
                        /* PRE (PWT clk prescaler) = 7 (Prescale by 2**7 = 128) */
                        /*     Count frequency = 20 MHz/128 = 156.25 KHz */
                        /* PWTEN (PWT enable) = 1 (PWT module enabled) */
                        /* PWTIE (PWT interrupt enable) = 0 (default) */
                        /* PRDYIE (PWT pulse width data ready interrupt ena)= 0 (default)*/
                        /* POVIE (PWT Counter overflow interrupt ena) = 0 (default) */
                        /* PWTSR (PWT Soft Reset) = 0 (default)*/
                        /* PWTDRY (PWT Pulse Width valid = 0 (default) */
                        /* PWTOV (PWT Counter OVerflow) = 0 (default, no overflow) */
  SIM_PINSEL1 &= ~SIM_PINSEL1_PWTIN0PS_MASK; /* Map PWT to pin PTD5 (default) */
}

void pulse_width_timer_PWT (void) {
  if (1==((PWT_R1 & PWT_R1_PWTRDY_MASK)>>PWT_R1_PWTRDY_SHIFT)) { /* If pulse with ready */
    PWT_R1 &= ~PWT_R1_PWTRDY_MASK;       /* Clear flag: read reg then write 0 to PWTRDY */
    PulseWidth = (PWT_R2 & PWT_R2_NPW_MASK) >> PWT_R2_NPW_SHIFT; /* Read pulse width */
           /* Pulse Width will be 19531 if connected to FTM2_ch1 */
  }
}
```

2.4.3.4 clocks.c

```
void init_clks_FEE_40MHz(void) {     /* FLL Enabled with External clock */
  OSC_CR = 0x96;       /* High range & gain; select osc */
                       /* OSCEN =1 ; OSC module enabled */
                       /* OSCSTEN = 0; OSC clock disabled in stop mode */
                       /* OSCOS = 1; OSC clcok source is selected */
                       /* RANGE = 1; High freq range of 4-24 MHz */
                       /* HGO = 1; High-gain mode */
  while ((OSC_CR & OSC_CR_OSCINIT_MASK) == 0); /* Wait until oscillator is ready*/
  ICS_C2 = 0x20;      /* BDIV div by 2; use default until dividers configured*/
                      /* LP = 0; FLL is not disabled in bypass mode */
  ICS_C1 = 0x18;      /* 8 Mhz ext ref clk/256 is source to FLL */
                      /* CLKS = 0; Output of FLL is selected (default) */
                      /* RDIV = 3; ref clk prescaled by 256 with RANGE=0 */
                      /* IREFS = 0; ext clk source selected */
                      /* IRCLKEN = 0; ICSIRCLK inactive */
```

```
                       /* IREFSTEN = 0; Int ref clk disabled in Stop mode */
  while ((ICS_S & ICS_S_IREFST_MASK) == 1);    /* Wait for external source selected */
  while ((ICS_S & ICS_S_LOCK_MASK) == 0);      /* Wait for FLL to lock */
  SIM_CLKDIV = 0x01100000; /* OUTDIV1 = 0; Core/sysclk is ICSOUTCLK div by 1 */
                           /* OUTDIV2 = 1  bus/flash is OUTDIV1/2 */
                           /* OUTDIV3 = 1; FTMs, PWT is ICSOUTCLK div by 2 */
  ICS_C2 = 0x00;           /* BDIV div by 1- increases bus/flash freq */
 }
```

2.5 Analog-to-Digital Conversion (ADC)

2.5.1 Description

Summary: The ADC is initialized for continuous conversion for two channels. One channel (AD10) connects to a potentiometer on the Freedom and evaluation board. The results are scaled to 0 to 5000 mV. On the Freedom and evaluation board, three LEDs are used to indicate the conversion result range.

Table 7. ADC example LED colors for voltage input from potentiometer

Scaled conversion result	LED Illuminated
3750 – 5000 mV	Red
2500 – 3750 mV	Green
1250 – 2500 mV	Blue
0 – 1250 mV	None

The other ADC input converted in the example is channel 11 (AD11), which connects to input PTC3. On the evaluation board, this input can be jumpered to ground if it is desired to read a value other than a floating input..

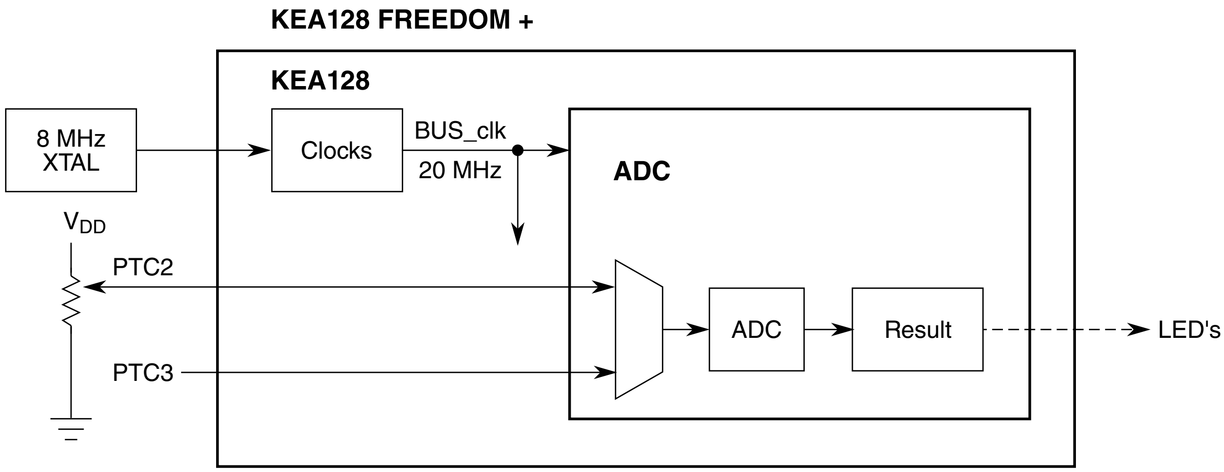

Figure 13. ADC example block diagram

2.5.2 Design steps

- Initialize clocks to FEE mode, 40 MHz system clock, 20 MHz TIMER_CLK
- Initialize GPIO outputs to LEDs on Freedom + evaluation board

- Initialize ADC module for continuous conversion, SW trigger and channels 10, 11 enabled to ADC
- Loop:
  - Issue ADC conversion command for AD10 (pin PTC2 to pot)
  - Wait for conversion complete flag. When conversion is complete:
    - Read result and convert to mV
    - Illuminate LED per voltage range
  - Issue ADC conversion command for AD11 (Pin PTC3)
  - Wait for conversion complete flag. When conversion is complete:
    - Read result, converted to mV. (It will be 0 mv if the pin is connected to ground, otherwise it will read a floating input value)
  - Increment counter

2.5.3 Code

2.5.3.1 main.c

```
#include "derivative.h"
#include "ADC.h"
#include "clocks.h"
#define PTE7  7                          /* Port PTE7 output to blue LED */
#define PTH0 24                          /* Port PTH0 output to red LED */
#define PTH1 25                          /* Port PTH1 output to green LED */

uint32_t adcResultInMv = 0;

int main(void){
  int counter = 0;
  int j;

  init_clks_FEE_40MHz();                /* KEA128 8MHz xtal: core 40 MHz, bus 20MHz */
  GPIOB_PDDR |= 1<<PTE7 | 1<< PTH0 | 1<<PTH1;       /* Output ports */
  GPIOB_PIDR &= 1<<PTE7 | 1<< PTH0 | 1<<PTH1;       /* Disable inputs (default) */

  init_ADC();      /* Init. ADC: Single conversion, SW trigger; enable adc chans 10 & 22 */

  for(;;) {

    convertAdcChan(10);                 /* Convert Channel AD10 to pot on EVB */
    while(adc_complete()==0){}          /* Wait for conversion complete flag */
    adcResultInMv = read_adc_chx();     /* Get channel's conversion results in mv */
    if (adcResultInMv > 3750) {         /* If result > 3.75V */
      GPIOB_PSOR |= 1<<PTE7 | 1<<PTH1;  /* turn off blue, green LEDs */
      GPIOB_PCOR |= 1<<PTH0;            /* turn on red LED */
    }
    else if (adcResultInMv > 2500) {    /* If result > 2.5V */
      GPIOB_PSOR |= 1<<PTE7 | 1<<PTH0;  /* turn off blue, red 2 LEDs */
      GPIOB_PCOR |= 1<<PTH1;            /* turn on green LED */
    }
    else if (adcResultInMv > 1250) {    /* If result > 1.25V */
      GPIOB_PSOR |= 1<<PTH0 | 1<<PTH1;  /* turn off red, green LEDs */
      GPIOB_PCOR |= 1<<PTE7;            /* turn on blue LED */
    }
    else {
      GPIOB_PSOR |= 1<<PTE7 | 1<< PTH0 | 1<<PTH1; /* Turn off all LEDs */
    }

    convertAdcChan(11);                 /* Convert Channel AD11 (pin PTC3) */
    while (adc_complete()==0){}         /* Test conversion complete flag */
    adcResultInMv = read_adc_chx();     /* Get channel's conversion results in mv */
```

```
    counter++;
  }
}
```

2.5.3.2 adc.c

```
#include "derivative.h" /* include peripheral declarations SKEAZ128M4 */
#include "ADC.h"

uint16_t adcResult = 0;                  /* ADC conversion result */

void init_ADC(void)  {
  SIM_SCGC |= SIM_SCGC_ADC_MASK;         /* Enable bus clock to ADC module */
  ADC_SC1 = 0x1F;             /* Disable module (default state) */
                              /* AIEN = 0: Interrupts disabled */
                              /* ADCO = 0: Continuous conversions disabled */
                              /* ADCH = 1F: Module disabled */
  ADC_APCTL1 =  0x00000C00;   /* Enable ADC channels 10 (PTC2), 11 (PTC3) */
  ADC_SC3 = 0x00000005;       /* Select ADCACLK, no divide, 10 bit conversion */
                              /* ADLPC = 0 (default): hi speed config */
                              /* ADIV = 0 (default): clock rate = input clock/1 */
                              /* ADLSMP = 0 (default): short sample time */
                              /* MODE = 1: 10 bit conversion */
                              /* ADICLK= 1:Bus clock/2 source */
  ADC_SC2 =0x00000000;        /* SW trigger, default ref pins, no compare */
                              /* ADTRG = 0 (default): SW Trigger */
                              /* ACFE = 0 (default):  compare function disabled */
                              /* REFSEL = 0 (default): default ref volt pin pair */
}

void convertAdcChan(uint16_t adcChan) {
  ADC_SC1 &= ~ADC_SC1_ADCH_MASK;        /* Clear any prior ADCH bits*/
  ADC_SC1 |= ADC_SC1_ADCH(adcChan);     /* Specify next channel for conversion */
}

uint8_t adc_complete(void)  {
  return ((ADC_SC1 & ADC_SC1_COCO_MASK)>>ADC_SC1_COCO_SHIFT);
                                      /* Return value of Conversion Complete flag */
}

uint32_t read_adc_chx(void)  {
  adcResult = ADC_R;       /* Read ADC conversion result (clears COCO flag) */
  return  (uint32_t) ((5000*adcResult)/0x3FF); /* Convert result to mv for 0-5V */
}
```

2.5.3.3 clocks.c

```
void init_clks_FEE_40MHz(void) {     /* FLL Enabled with External clock */
  OSC_CR = 0x96;       /* High range & gain; select osc */
                       /* OSCEN =1 ; OSC module enabled */
                       /* OSCSTEN = 0; OSC clock disabled in stop mode */
                       /* OSCOS = 1; OSC clcok source is selected */
                       /* RANGE = 1; High freq range of 4-24 MHz */
                       /* HGO = 1; High-gain mode */
  while ((OSC_CR & OSC_CR_OSCINIT_MASK) == 0); /* Wait until oscillator is ready*/
  ICS_C2 = 0x20;      /* BDIV div by 2; use default until dividers configured*/
                      /* LP = 0; FLL is not disabled in bypass mode */
  ICS_C1 = 0x18;      /* 8 Mhz ext ref clk/256 is source to FLL */
                      /* CLKS = 0; Output of FLL is selected (default) */
                      /* RDIV = 3; ref clk prescaled by 256 with RANGE=0 */
                      /* IREFS = 0; ext clk source selected */
                      /* IRCLKEN = 0; ICSIRCLK inactive */
                      /* IREFSTEN = 0; Int ref clk disabled in Stop mode */
  while ((ICS_S & ICS_S_IREFST_MASK) == 1);    /* Wait for external source selected */
  while ((ICS_S & ICS_S_LOCK_MASK) == 0);      /* Wait for FLL to lock */
  SIM_CLKDIV = 0x01100000; /* OUTDIV1 = 0; Core/sysclk is ICSOUTCLK div by 1 */
```

```
                           /* OUTDIV2 = 1  bus/flash is OUTDIV1/2 */
                           /* OUTDIV3 = 1; FTMs, PWT is ICSOUTCLK div by 2 */
  ICS_C2 = 0x00;           /* BDIV div by 1- increases bus/flash freq */
}
```

2.6 UART

2.6.1 Description

Summary: The UART module is configured for 9600 baud, 8 bits, 1 stop bit and no parity. Software then transmits and receives characters.

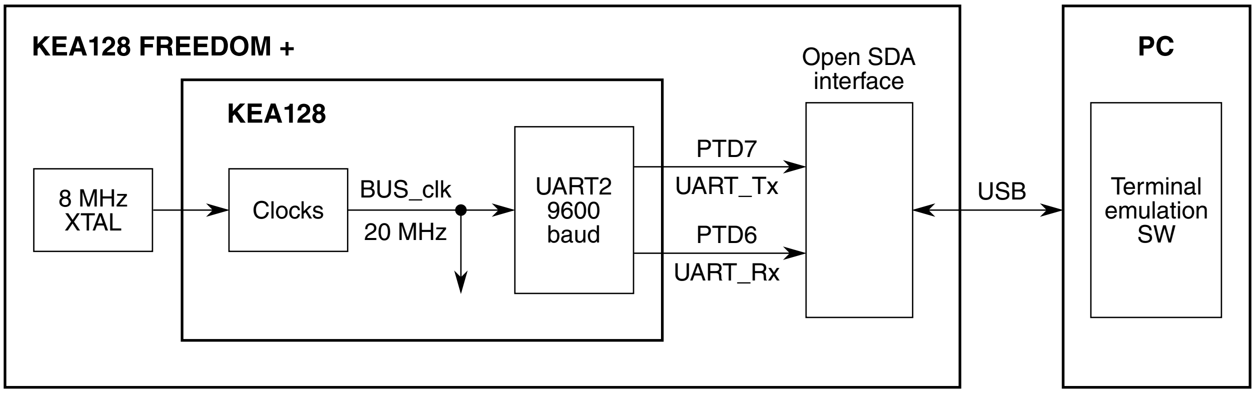

Figure 14. UART example block diagram

For this example the UART connects to an Open SDA Interface where the USB connection is used for both the terminal and debugger functions. Separate software on a personal computer, such as Windows HyperTerm, is needed to communicate.

The BUS_CLK is the source clock for UART modules, configured here for 20 MHz. BUS_CLK is divided to get the desired baud rate by 16 times the Baud Rate Divisor, SBR. SBR number is contained in registers UARTx_BDH and UARTx_BDL. Baud rate for this configuration is:

```
Baud Rate = BUS_CLK / 16 / Baud Rate Divisor (SBR)
          = 20 MHz / 16 / 130 ~= 9600 baud
```

The following scope shot shows the letter “H” being transmitted. The ASCII value for “H” is 0x48 or 0b0100_1000. Bits are transmitted LSB first. After the START bit (0), the bits are 0b0001_0100 which is “H” sent LSB first. At the end of the 8 data bits, there is a STOP bit of 1.

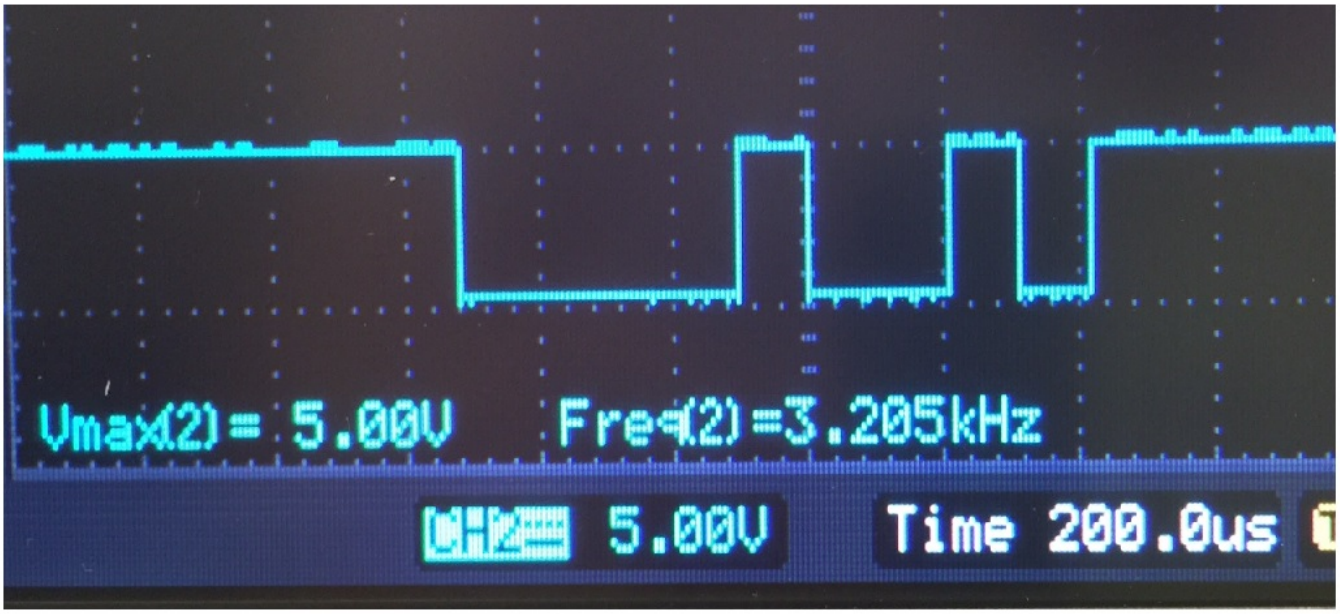

Figure 15. UART example scope shot of letter "H" transmission

Access to the UART port can be done using the KEA128 Freedom + board Open SDA Interface circuit. Using a common terminal emulation like TeraTerm set the serial port over USB to 9600 baud, 1 stop, no parity.

2.6.2 Design steps

- Initialize clocks to FEE mode, 40 MHz system clock, 20 MHz TIMER_CLK
- Initialize UART2 9600 baud, 1 stop, no parity, no interrupts.
- Transmit a character string
- Loop:
  - Transmit a prompt character
  - Wait for user input of on character
  - Receive user character and echo it back

2.6.3 Code

2.6.3.1 main.c

```
#include "derivative.h" /* include peripheral declarations SKEAZ128M4 */
#include "UART.h"
#include "clocks.h"

int main(void)
{
  int counter = 0;

  init_clks_FEE_40MHz();    /* KEA128 clks FEE, 8MHz xtal: core 40 MHz, bus 20MHz */
  init_UART();                /* Initialize UART .... */
  transmit_string("Running UART example\n\r"); /* Transmit string, new line, return */
  for(;;) {
    transmit_char('>');       /* Transmit a "prompt" type character */
    recieve_and_echo_char();  /* Wait for an input char, read it & echo it back */
```

```
    counter++;                /* Loop counter */
  }
}
```

2.6.3.2 UART.c

```
#include "derivative.h" /* include peripheral declarations SKEAZ128M4 */
#include "derivative.h"
#include "UART.h"
/* Initialize UART at Baud Rate = 9600, 1 stop bit, 8 bit format, no parity */
void init_UART(void)  {
  SIM_SCGC |=  SIM_SCGC_UART2_MASK;  /* Enable bus clock (20MHz) in UART2 */
  UART2_BDH = 0;                     /* One stop bit; upper baud divisor bits = 0 */
  UART2_BDL = 130;                   /* For 9600 baud: baud divisor=20M/9600/16 = ~130 */
  UART2_C1  = 0;                     /* Initialize control bits for communication: */
                                     /* M (9, 8 bit select) = 0 (default, 8 bit format) */
                                     /* PE (Parity Enable) = 0 (default, no parity) */
                                     /* UARTSWAI (UART in wait mode)=0 (default, no stop)*/
                                     /* WAKE (Recvr Wakeup Method)=0 (default idle-line) */
                                     /* ILT (Idle Line Type Select) = 0 */
                                     /*     (default, bit counts after start) */
  UART2_C2 = 0x0C;                   /* Enable Tx, Rx. No IRQs, Rx in standby, break char*/
  SIM_PINSEL1 &= ~SIM_PINSEL1_UART2PS_MASK; /* UART2PS=0 (default); */
                                            /* UART2 Pin Selection Tx PTD7,Rx PTD6 */
}
/* Function to Transmit single Char */
void transmit_char(char send) {
  while((UART2_S1 & UART_S1_TDRE_MASK)==0); /* Wait for transmit buffer to be empty */
  (void)UART2_S1;                    /* Read UART2_S1 register to clear TDRE */
  UART2_D=send;                      /* Send data */
}
/* Function to Transmit whole string */
void transmit_string(char data_string[])  {
  int i=0;
  while(data_string[i] != '\0')  {   /* Send chars one at a time */
    transmit_char(data_string[i]);
    i++;
  }
}
/* Function to Receive single Char */
char receive_char(void) {
  char recieve;
  while((UART2_S1 & UART_S1_RDRF_MASK)==0);  /* Wait for received buffer to be full */
  (void) UART2_S1;          /* Read UART2_S1 register to clear RDRF (after reading data) */
  recieve= UART2_D;         /* Read received data*/
  return recieve;
}
/* Function to echo the received char back to the Sender */
void recieve_and_echo_char(void)  {
  char send = receive_char();        /* Receive Char */
  transmit_char(send);               /* Transmit same char back to the sender */
  transmit_char('\n');               /* New line */
  transmit_char('\r');               /* Return */
}
```

2.6.3.3 clocks.c

```
void init_clks_FEE_40MHz(void) {     /* FLL Enabled with External clock */
  OSC_CR = 0x96;       /* High range & gain; select osc */
                       /* OSCEN =1 ; OSC module enabled */
                       /* OSCSTEN = 0; OSC clock disabled in stop mode */
                       /* OSCOS = 1; OSC clcok source is selected */
                       /* RANGE = 1; High freq range of 4-24 MHz */
                       /* HGO = 1; High-gain mode */
  while ((OSC_CR & OSC_CR_OSCINIT_MASK) == 0); /* Wait until oscillator is ready*/
```

```
  ICS_C2 = 0x20;       /* BDIV div by 2; use default until dividers configured*/
                       /* LP = 0; FLL is not disabled in bypass mode */
  ICS_C1 = 0x18;       /* 8 Mhz ext ref clk/256 is source to FLL */
                       /* CLKS = 0; Output of FLL is selected (default) */
                       /* RDIV = 3; ref clk prescaled by 256 with RANGE=0 */
                       /* IREFS = 0; ext clk source selected */
                       /* IRCLKEN = 0; ICSIRCLK inactive */
                       /* IREFSTEN = 0; Int ref clk disabled in Stop mode */
  while ((ICS_S & ICS_S_IREFST_MASK) == 1);    /* Wait for external source selected */
  while ((ICS_S & ICS_S_LOCK_MASK) == 0);      /* Wait for FLL to lock */
  SIM_CLKDIV = 0x01100000; /* OUTDIV1 = 0; Core/sysclk is ICSOUTCLK div by 1 */
                           /* OUTDIV2 = 1  bus/flash is OUTDIV1/2 */
                           /* OUTDIV3 = 1; FTMs, PWT is ICSOUTCLK div by 2 */
  ICS_C2 = 0x00;           /* BDIV div by 1- increases bus/flash freq */
 }
```

3 Startup code

This section is intended to give a high level map from reset to main with a compiler's startup files and functions. Often clocks can be configured by defines in a header file, as in S32 Design Studio.

3.1 S32 Design Studio, flash target

Table 8. S32 Design Studio

	startup_SKEAZ128M4.s	system_SKEAZ128M4.h	system_SKEAZ128M4.c	ARM libraries
1	__isr_vector - Defines stack & interrupt addresses such as Reset_Handler - Uses symbols from link file	—	—	—
2	Reset_Handler: - Mask, disable & clear interrupts - SystemInit	—	—	—
3	—	DISABLE_WDOG=1 (default) CLOCK_SETUP = 0 (default) - FEI mode - ICS ref clk ~= 32 kHz - Core clk = 20.97 MHz - Bus clock = 20.97 MHz	SystemInit: - disable watchdog - clock setup (FEI mode) - Update system prescalers (OUTDIV1) - Intializes ICS, OSC	—
4	- Unmask all interrupts	—	—	—

Table continues on the next page...

Table 8. S32 Design Studio (continued)

	startup_SKEAZ128M4.s	system_SKEAZ128M4.h	system_SKEAZ128M4.c	ARM libraries
	- ROM to RAM variable initialization			
5	Branch to __START	—	—	—
—	—	—	—	__START: Various compiler initializations such as initializing RAM variables
6	—	—	—	Branch to main

4 KEA header file usage

Table 9. Configuring registers and bit fields KEA Header File Example Uses

Example	Union of Structures (MPC5xxx)	KEA Macros
Initialize register	*MODULE.REG.R = value*; *Example(s):* SIUL.PCR[40].R = 0x1234;	MODULE_REG = value; *Example(s)*: ICS_C3 = 0x90; FTM2_C0SC = 0x68; /* FTM2 ch 0 SC*/
Initialize bit field	*MODULE.REG.B.FIELD = value*; *Example(s)*: SIUL.PCR[40].B.PA = 3;	*MODULE_REG &= ~MOD_REG_FIELD_MASK*; *MODULE_REG |= MOD_REG_FIELD(value)*; *Example(s)*: ADC_SC1 &= ~ADC_SC1_ADCH_MASK; // clear field ADC_SC1 |= ADC_SC1_ADCH(adcChan); // init field
Set single bit in register	*MODULE.REG.B.FIELD = 1*; *Example(s)*: SIUL.PCR[40].B.OBE = 1;	*MODULE_REG |= MOD_REG_FIELD_MASK*; *-or- MODULE_REG |= 1<<CONSTANT*; *Example(s)*: SIM_SCGC |= SIM_SCGC_FTM2_MASK; GPIOA_PDDR |= 1<<PTC0;
Clear single bit in register	*MODULE.REG.B.FIELD = 0*; *Example(s)*: SIUL.PCR[40].B.OBE = 0;	*MODULE_REG &= ~(1<<CONSTANT); or* *-or- MODULE_REG &= ~MOD_REG_FIELD_MASK;* *Example(s):*

Table 9. Configuring registers and bit fields KEA Header File Example Uses

Example	Union of Structures (MPC5xxx)	KEA Macros
		GPIOA_PDDR &= ~(1<<PTC0); I2C_C1 & = ~I2C_C1_TX_MASK

Table 10. Using the header files to read and test registers

Example	Union of Structures (MPX5xxx)	KEA Macros
Read Bit	*x = MODULE.REG.B.FIELD*; *Example(s)*: x = SIUL.GPDI[5].B.PDI;	*x = ((MODULE_REG>>CONSTANT) & 1);* *Example(s)*: x = (GPIOA_PIDR >> PTD0) & 1;
Read Field	*x = MODULE.REG.B.FIELD*; *Example(s)*: x = SIUL.GPDI[5].B.PDI;	*x = (MODULE_REG & MOD_REG_FIELD_MASK) >> MOD_REG_FIELD_SHIFT* *Example(s)*: x = (I2C_A1 & I2C_A1_AD_MASK) >> I2C_A1_AD_SHIFT;

5 Revision History

Rev. No.	Date	Substantive Change(s)
0	Feb 2016	Initial version

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number AN5213
Revision 0, Feb 2016