CG2271 Cheat Sheet

GPIO Configuration

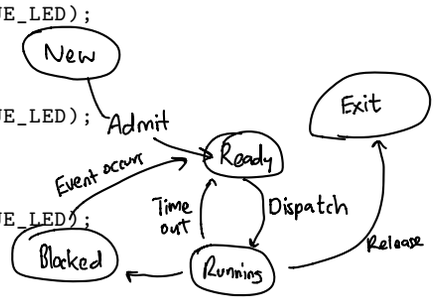
```c
void InitGPIO(void) {
  // Enable Clock to PORTB and PORTD
  SIM->SCGC5 |=
    ((SIM_SCGC5_PORTB_MASK) | (SIM_SCGC5_PORTD_MASK));

  // Configure MUX settings to make all 3 pins GPIO
  PORTB->PCR[RED_LED] &= ~PORT_PCR_MUX_MASK;
  PORTB->PCR[RED_LED] |= PORT_PCR_MUX(1);
  PORTB->PCR[GREEN_LED] &= ~PORT_PCR_MUX_MASK;
  PORTB->PCR[GREEN_LED] |= PORT_PCR_MUX(1);
  PORTD->PCR[BLUE_LED] &= ~PORT_PCR_MUX_MASK;
  PORTD->PCR[BLUE_LED] |= PORT_PCR_MUX(1);

  // Set Data Direction Registers for PortB and PortD
  PTB->PDDR |= (MASK(RED_LED) | MASK(GREEN_LED));
  PTD->PDDR |= MASK(BLUE_LED);
}

void turn_on_blue() {
  PTD->PCOR |= MASK(BLUE_LED);
}

void turn_off_blue() {
  PTD->PSOR |= MASK(BLUE_LED);
}
```



ISR Configuration

```c
void InitSwitch() {
  // Enable clock for port D
  SIM->SCGC5 |= SIM_SCGC5_PORTD_MASK;

  // Set interrupt on falling edge
  PORTD->PCR[SW_POS] &= ~PORT_PCR_MUX_MASK;
  PORTD->PCR[SW_POS] &= ~PORT_PCR_IRQC_MASK;
  PORTD->PCR[SW_POS] |=
    PORT_PCR_MUX(1) |
    PORT_PCR_PS_MASK |
    PORT_PCR_PE_MASK |
    PORT_PCR_IRQC(0x0a);

  // Set pin to input
  PTD->PDDR &= ~MASK(SW_POS)

  // Enable interrupts
  NVIC_SetPriority(PORTD_IRQn, 128);
  NVIC_ClearPendingIRQ(PORTD_IRQn);
  NVIC_EnableIRQ(PORTD_IRQn);
}
```

Non-Atomic Data: >32 bits, takes more than 1 instruction to update

Solution: Disable preemption

```c
uint32_t m = __get_PRIMASK();
__disable_IRQ();
// Do Stuff
__set_PRIMASK(m);
```

Interrupt Handler

```c
void PORTD_IRQHandler(void) {
  // clear pending interrupts
  NVIC_ClearPendingIRQ(PORTD_IRQn);

  if ((PORTD->ISFR & MASK(SW_POS))) {
    button_pressed = 1;
  }

  // clear status flags
  PORTD->ISFR = 0xffffffff;
}
```

PWM Initialization

```c
void setFreq(int freq) {
  // 375k = clk spd / prescaler
  int width = 375000/freq;

  TPM1->MOD = width-1;
  TPM1_C0V = width/2;
}

void initPWM(int freq) {
  // Enable clk to port B
  SIM_SCGC5 |= SIM_SCGC5_PORTB_MASK;

  // Set Pins 0 and 1 to be connected to the TPM module
  PORTB->PCR[PTB0_Pin] &= ~PORT_PCR_MUX_MASK;
  PORTB->PCR[PTB0_Pin] |= PORT_PCR_MUX(3);
  PORTB->PCR[PTB1_Pin] &= ~PORT_PCR_MUX_MASK;
  PORTB->PCR[PTB1_Pin] |= PORT_PCR_MUX(3);

  // (pg 208) Powers on TPM1 module
  SIM->SCGC6 |= SIM_SCGC6_TPM1_MASK;

  // (pg 195) Resets the current state of the mask
  SIM->SOPT2 &= ~SIM_SOPT2_TPMSRC_MASK;
  // Configures TPM to use MCG FLL clock (pg 367)
  // MCG = Main Clock Generator
  // PLL = Phase Locked Loop
  // It is 48 MHz (?)
  SIM->SOPT2 |= SIM_SOPT2_TPMSRC(1);

  // (pg 554) Sets the modulo value
  setFreq(freq);

  // (pg 553) Resets clk mode & prescaler
  TPM1->SC &= ~((TPM_SC_CMOD_MASK) | (TPM_SC_PS_MASK));
  // Set clk mode to increment on internal clk
  // prescalar = 128
  TPM1->SC |= (TPM_SC_CMOD(1) | TPM_SC_PS(7));

  // (pg 553) Set to 0 -> Up-counting mode -> Edge aligned PWM
  TPM1->SC &= ~(TPM_SC_CPWMS_MASK);

  // (pg 556) Reset old masks
  TPM1_C0SC &= ~(TPM_CnSC_ELSB_MASK | TPM_CnSC_ELSA_MASK | TPM_CnSC_MSB_MASK | TPM_CnSC_MSA_MASK);
  // Set edge aligned PWM, high true pulses
  TPM1_C0SC |= TPM_CnSC_ELSB(1) | TPM_CnSC_MSB(1);
}
```

Our board: FRDM-KL25Z

Freescale Kinetis MKL25Z128VLK4 ↖ microcontroller! processor!

- 48 MHz clk

32 bit ARM cortex M0+ Core

UART

```c
void initUART2(uint32_t baud_rate) {
  uint32_t divisor, bus_clock;

  // Enable power to UART and port E
  SIM->SCGC4 |= SIM_SCGC4_UART2_MASK;
  SIM->SCGC5 |= SIM_SCGC5_PORTE_MASK;

  // Set pin to UART mode (alt mode)
  PORTE->PCR[UART_TX_PORTE22] &= ~PORT_PCR_MUX_MASK;
  PORTE->PCR[UART_TX_PORTE22] |= PORT_PCR_MUX(4);
  PORTE->PCR[UART_RX_PORTE23] &= ~PORT_PCR_MUX_MASK;
  PORTE->PCR[UART_RX_PORTE23] |= PORT_PCR_MUX(4);

  // Clear TX and RX settings
  UART2->C2 &= ~((UART_C2_TE_MASK) | (UART_C2_RE_MASK));

  // Set baud rate
  bus_clock = (DEFAULT_SYSTEM_CLOCK) / 2;
  divisor = bus_clock / (baud_rate * 16);
  UART2->BDH = UART_BDH_SBR(divisor >> 8);
  UART2->BDL = UART_BDL_SBR(divisor);

  UART2->C1 = 0;
  UART2->S2 = 0;
  UART2->C3 = 0;
  // Enable TX and RX
  UART2->C2 |= ((UART_C2_TE_MASK) | (UART_C2_RE_MASK));

  // Enable interrupts
  NVIC_SetPriority(UART2_IRQn, UART2_INT_PRIO);
  NVIC_ClearPendingIRQ(UART2_IRQn);
  NVIC_EnableIRQ(UART2_IRQn);

  // Set RX interrupt
  UART2->C2 |= UART_C2_RIE_MASK;
}

// Not used in favour of interrupts
uint8_t UART2_Receive_Poll(void) {
  while(!(UART2->S1 & UART_S1_RDRF_MASK));
  return UART2->D;
}

volatile int data_received = 0;
volatile int data = 0;
```

Shardeen Modeth
Mohideen Imran Khan

```c
void UART2_IRQHandler(void) {
  if (UART2->S1 & UART_S1_RDRF_MASK) {
    // received a character
    data_received = 1;
    data = UART2->D;
  }
}
```

Processes

```c
void led_red_thread (void *argument) {
  for (;;) {
    led_control(RED, led_on);
    delay(0x80000);
    led_control(RED, led_off);
    delay(0x80000);
  }
}

void led_green_thread (void *argument) {
  for (;;) {
    led_control(GREEN, led_on);
    delay(0x80000);
    led_control(GREEN, led_off);
    delay(0x80000);
  }
}

int main (void) {

  // System Initialization
  SystemCoreClockUpdate();
  InitGPIO();
  offRGB();

  // Initialize CMSIS-RTOS
  osKernelInitialize();
  // Create application red led thread
  osThreadNew(led_red_thread, NULL, NULL);
  // Create application green led thread
  osThreadNew(led_green_thread, NULL, NULL);
  // Start thread execution
  osKernelStart();
  for (;;) {}
}
```

Compiled by **Mohideen Imran Khan**.
Template from http://wch.github.io/latexsheet/

RTC Scheduler
- Uses a table to store info of task (Period, Release time, ready)
- Can also be pre-emptive

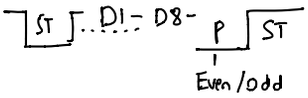
Microcontroller vs Microprocessor:
- Controller has peripherals such as clock generator, I/O, timers, RAM, ROM

What is RTOS?
- Guarantees maximum response time for each task.

RAM requirement:
RTC Static / Dynamic → ∑static + max(func. stack)
Preemptive → ∑ static + ∑ function stack

Communication
- Parallel bus, shared by peripherals
- Full duplex Serial bus – send & receive same time
- Half duplex → send & receive on shared line

Asynchronous Rx & Tx use own clock



Use 1 queue to store data to be sent, 1 to store received data.
Load / Unload in ISRs.

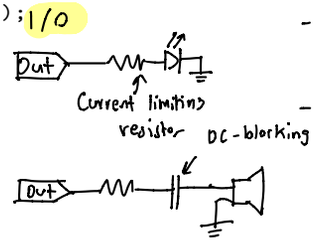
CPU architectures
Load/Store → RISC
Register/Memory → CISC
Memory wall → ?

PSR: Program Status Register
APSR: NZCV flags
IPSR: Stores exception no. of ISR
EPSR: Thumb state

OS Structure:
Monolithic: Single program with kernel and all OS services.
- Fast, riskier

Microkernel: Simple kernel, User-space servers.
- More memory, slower, easy to develop.

I/O



- Instructions are little Endian
- Thumb-2 instructions
  - Mostly 16 bits long
  - Half-word aligned
  - Odd PC

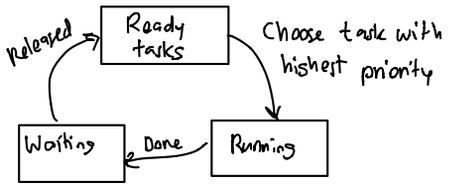
PCB: Contains info about processes

Concurrency:
Interrupts: run a function everytime some external event occurs.
Enough for simple systems.

Task scheduler:
- Use software to schedule CPU time.

Static (Cyclic Executive)
- Run tasks ABCD in same order in a loop.
- Max delay is sum of T_{task}

Dynamic RTC = Run to completion (no pre-emption)



How to detect hardware event?
- Polling
- Interrupt
  1. Finish current instruction
  2. Push context onto stack (8 registers total)
  3. Switch to handler/priviledged mode, use MSP. (other mode is thread mode)
  4. Load PC with addr. of exception handler
  5. Load LR with EXC-RETURN code → Which SP to revert to, Which mode
  6. Load IPSR
  7. Start execution

Dynamic Pre-emptive
- Higher priority task can interrupt a lower priority task

Cyclic Executive with interrupts
Back ground: A, B, C
Foreground: ISR: do urgent task
Main code uses Round Robin Scheduling

Vector table: contains address of every ISR

- **Volatile:** Force CPU to reload variable from RAM every time, don't reuse register value

Do we run tasks in the same order each time?
- Yes: Static schedule
- No: Dynamic schedule

Can one task pre-empt another?
- Yes: Preemptive
- No: Non-preemptive

Time definitions: