

```
;IMPORTANT!!!!!!!!!!!!!!
;This is a BETA version of Firmware that has not been fully tested.
;PLEASE keep this in mind and use this firmware as REFERENCE.
```

```
=====
;
;   MAIN ASM FILE FOR FRAMEWORKS
;=====
;   Change History
;   -----
;   Revision:      0.1
;   Created DGW   November 1999
;   New Code
;   Issue Date:   DRAFT
;=====
;   Framewks.asm
;
;   This file contains the Frameworks for:-
;   CY7C637xx
;=====
```

CPU 63743

```
include "usblregs.h"
include "ram.h"
```

```
=====
;
;   Interrupt Vectors
;=====
```

org 00h

```
    jmp reset           ; Reset vector
    jmp bus_reset      ; USB Reset interrupt
    jmp Isr128         ; 128us interrupt
    jmp Isr1ms         ; 1024ms interrupt
    jmp IsrEp0         ; endpoint 0 interrupt
    jmp IsrEp1         ; endpoint 1 interrupt
    jmp IsrEp2         ; endpoint 2 interrupt
    jmp IsrSPI         ; SPI interrupt
    jmp IsrCaptureA    ; Capture timer A interrupt
    jmp IsrCaptureB    ; Capture timer B interrupt
    jmp IsrGpio        ; general purpose I/O interrupt
    jmp IsrWakeup      ; Wakeup_ISR or resume interrupt
    ; jmp IsrCext      ; Wakeup_ISR or resume interrupt
```

```
=====
;
; reset processing
;=====
;
; Set all SFRs to their initial states
; Clear ram
;=====
```

reset:

```
    mov A, EP2_FIFO           ;move data stack pointer
    swap A, dsp               ;so it does not write over USB FIFOs

    mov a, VREG_EN_B
    iowr USB_SCR              ; enable USB
    mov a, 80h
    iowr USB_DA               ; enable USB address
    mov a, 0Fh
    iowr EP0_MODE
    mov a, 06h
    iowr CLOCK_CONFIG
    mov a, ACKIN
    iowr EP1_MODE             ;initialize endpoint 1 mode to accept in's
    mov a, ACKOUT
    iowr EP2_MODE

    mov a, 01h
```

```

iowr GIER
mov a, 07h
iowr EP_IE
ei

iord SCR
and a, LVR_BROUT_B;RESET_MASK
cmp a, LVR_BROUT_B      ; test for POR(Low Voltage or Brown out) Reset
jz por_reset

and A, BUS_INT_EVENT      ; test for USB Reset
jz bus_reset

jmp wdog_reset

;=====
;
; 128us Interrupt
;
;=====
;
; {
; If (the previous 128us interrupt has been missed)
;   {make appropriate corrections}
; If (the timer has rolled over)
;   {increment timer msb}
; Return from Interrupt
; }
;
;=====

Isrl28:
  push a

  iord TIMER_LSB          ; check that we haven't missed an interrupt
  xor a, [timer_sync]    ; timer_sync saves the msb of timer as a
  and a, BIT7            ; check against missed 128us interrupts.
  jnz in_sync            ;

  or a, [timer_sync]     ; missed, so make corrections - note a=0 because of jnz
  jz in_sync             ; if 0 then we didn't miss an overflow
  inc [timer_msb]        ; if not we did, so correct missed overflow

in_sync:
  iord TIMER_LSB          ; check to see if the timer has overflowed
  and a, BIT7            ;
  mov [timer_sync], a    ;
  jnz .Reti              ;
  inc [timer_msb]        ; adjust the msb if it has

.Reti:
  mov a, (EP0_IE+EP1_IE+EP2_IE)
  iowr EP_IE              ; enable EP interrupt here because no longer in GIER
  mov a, [interrupt_mask] ;
  iowr GIER               ;
  pop a
  ei
  reti

;=====
;
; Isrlms
;
;=====
;
; {
; Increment task number
; Check and flag missed 128us interrupts
; If (no bus activity in last lms)
;   inc inactivity timer
; }
;
;=====

Isrlms:
  push a
  inc [task_number]

  mov a, 00h              ; flag lms event to main code

```

```

    mov [new_task], a          ;

    mov a, [timer_msb]        ; check that no 128us ISRs have been missed
    and a, (BIT0 + BIT1)      ; Bits 1 and 2 should always be zero,
    jz .CheckActivity         ; timer error should be checked and cleared after
    mov [timer_error], a      ; any time measurement.

.CheckActivity:
    iord USB_SCR
    and a, ACTIVITY_B
    jz .IncCtr

    iord USB_SCR
    and a, ~ACTIVITY_B
    iowr USB_SCR

    mov a, 0
    mov [suspend_counter], a
    jmp .Reti

.IncCtr:
    inc [suspend_counter]

.Reti:
    mov a, (EP0_IE+EP1_IE+EP2_IE)
    iowr EP_IE                ; enable EP interrupt here because no longer in GIER
    mov a, [interrupt_mask]   ;
    iowr GIER                 ;
    pop a
    reti

;=====
;
;   IsrEp0
;
;=====
;
;   {
;   }
;
;=====

IsrEp0:
    push a
    iord USB_DA                ; re-enable USB
    or a, 80h
    iowr USB_DA
    iord EP0_MODE              ; re-enable mode
    or a, 0Fh
    iowr EP0_MODE
    and a, EP0_SETUP_B        ; check if SETUP packet received
    jz .NotSetup              ;

    call DecodeRequest         ; parse SETUP packet
    jmp .Reti

.NotSetup:
    iord EP0_MODE              ;
    and a, EP0_OUT_B          ; ignore null packet interrupts
    jnz .Reti

    iord EP0_MODE              ; do not continue unless xfer of last paket is complete
    ;and a, EP0_ACK            ; note the Input Enable bit is set when the host ACKs
    ;jz .Reti

                                ; will need to add more checks here when set report and
                                ; get report support is added

    call ControlReadDataStage  ; continue sending control read data packets

.Reti:
    mov a, 0fh
    iowr EP0_MODE
    mov a, (EP0_IE+EP1_IE)
    iowr EP_IE                ; enable EP interrupt here because no longer in GIER
    mov a, [interrupt_mask]   ; enable endpoint zero interrupts
    iowr GIER                 ;
    pop a
    reti

```

```

;=====
;
;   IsrEpl
;
;=====
;
;   {
;   invert the data toggle bit
;   }
;
;=====

IsrEpl:
    push a

    iord EP1_COUNT
    xor a, DATA01_B
    or a, 01h
    iowr EP1_COUNT

    iord EP1_MODE
    mov a, 0fh
    iowr EP1_MODE
    mov a, (EP0_IE+EP1_IE)
    iowr EP_IE          ; enable EP interrupt here because no longer in GIER
    mov a, [interrupt_mask] ;
    iowr GIER          ;
    pop a
    reti

;=====
;
;   IsrGpio
;
;=====
;
;   {
;   }
;
;=====

IsrGpio:
    push a
    mov a, (EP0_IE+EP1_IE)
    iowr EP_IE          ; enable EP interrupt here because no longer in GIER
    mov a, [interrupt_mask] ;
    iowr GIER          ;
    pop a
    reti

;=====
;
;   process_reset
;
;=====
;
;   {
;   }
;
;=====

por_reset:
    mov a, ffh
    iowr P0_data          ;set all ports to output 1
    iowr P1_data

    mov a, 00h
    iowr P0_interrupt      ;disable gpio interrupts on all pins
    iowr P1_interrupt      ;
    iowr P0_Mode0          ;Setup all ports for Resistive
    iowr P1_Mode0          ;ie pullup states
    mov a, FFh            ;
    iowr P0_Mode1          ;
    iowr P1_Mode1          ;

    iowr WDT                ; clear wdt timer

```

```

    mov a, bottom_of_ram
    mov x, top_of_ram
ram_clr_loop:                ; clear all ram locations
    mov [x+0], a
    dec x
    jnc ram_clr_loop

    iord SCR
    and a, BUS_INT_EVENT
    jnz bus_reset

suspend_reset:
    iowr WDT
    jmp suspend_reset
    mov A, (RUN_B + SUSPEND_B)
    iowr SCR                ; suspend the mcu
    nop
    jmp suspend_reset        ; wait until real bus reset

wdog_reset:
    jmp suspend_reset        ;suspend and wait for USB Bus Reset

bus_reset:
    mov a, 80h
    iowr USB_DA
    mov A, RUN_B            ; clear all reset bits
    iowr SCR

    mov a, EP0_IE
    iowr EP_IE                ; enable EP interrupt here because no longer in GIER                ; setup
for enumeration
    mov A, ENUMERATE_MASK
    mov [interrupt_mask],A
    iowr GIER
    ei

;=====
;
;  project-specific initialisation
;
;=====
;
;  {
;  Do project-specific initialisation
;  }
;
;=====

UserReset:

;=====
;
;  main
;
;=====
;
;  {
;  reset watchdog timer
;  check for USB suspend condition
;  if (time to do next task)
;  {
;      if (device is not enumerated)
;          jump to pre-enumeration task
;      else if (task number out of range)
;          task number := 0
;      jump to task
;  }
;  }
;
;=====

main:
    iowr WDT                ; kick the dog
    call CheckSuspend

    and a, [new_task]        ; new_task is active if = 0
    jnz main                ; wait for next time slice
    inc [new_task]

```

```

mov a, configuration_status      ; for designs intended to power-up when
and a, ffh                      ; disconnected from the bus, this code
jz NotEnumerated                ; may not be appropriate.

mov a, [task_number]            ; jump to current tasklet
asl                             ; x2 because jmp is 2 instructions
cmp a, (end_task_table - task_table) ; check to ensure not out of range

jc .DoTheJump
mov a, 00h
mov [task_number], a           ; user may want to add own error recovery code here

.DoTheJump:
jacc task_table                ; execute the selected task

XPAGEOFF
task_table:
jmp Task0
jmp Task1
jmp Task2
jmp Task3
jmp Task4
jmp Task5
jmp Task6
jmp Task7
end_task_table:
XPAGEON

;=====
;
; Check Suspend
;
;=====
;
; {
; if (time since last usb activity is > suspend time)
; {
; set io ports into suspend state
; set interrupts into suspend state
; while (no bus activity)
; suspend
; restore interrupts
; restore io state
; }
; return
; }
;=====

CheckSuspend:
mov a, [suspend_counter]        ; check the suspend timer to see how long since last
sub a, SUSPEND_TIME             ; bus activity.
jc .Ret                          ;

mov a, 00h                      ; clear the suspend counter to avoid any risk we might
mov [suspend_counter], a        ; suspend again before the lms isr clears it on wakeup

iord P0_data                    ; save io port status
push a                          ;
iord P1_data                    ;
push a                          ;
iord P0_Mode0                   ;
push a                          ;
iord P1_Mode0                   ;
push a                          ;
iord P0_Mode1                   ;
push a                          ;
iord P1_Mode1                   ;
push a                          ;

mov a, ffh                      ; to ensure minimum current drain during suspend
iowr P0_data                    ; all ports must be logic 1 with pullup on unless
iowr P1_data                    ; externally driven.
mov a, 00h                      ;
iowr P0_Mode0                   ;
iowr P1_Mode0                   ;
mov a, ffh                      ;
iowr P1_Mode1                   ; user to modify these values depending on circuit
iowr P0_Mode1                   ;

```

```

iord GIER          ; set which interrupts may cause wake from suspend
and a, SUSPEND_MASK ; user may also want to change GPIO interrupt
iowr GIER          ; masking here

iord SCR          ; put the device into suspend
or a, SUSPEND_B   ;
iowr SCR          ;

nop               ; instruction the code wakes up to

pop a             ; Restore io port status
mov a, FFh
iowr P1_Model
pop a
mov a, FFh
iowr P0_Model
pop a
mov a, 00h
iowr P1_Mode0
pop a
mov a, 00h
iowr P0_Mode0
pop a
mov a, FFh        ;
iowr P1_data      ;
pop a
mov a, FFh        ;
iowr P0_data      ;

mov a, (EP0_IE+EP1_IE)
iowr EP_IE        ; enable EP interrupt here because no longer in GIER
mov a, [interrupt_mask] ;
iowr GIER         ;
ei

.Ret:
ret

;=====
;
; Not Enumerated
;
;=====
;
; This module is run if enumeration is incomplete
;
;=====

NotEnumerated:

    jmp main

;=====
;
; Task 0
;
;=====
;
; {
; read Buttons
;
; }
;
;=====

Task0:
    iord Buttons_port
    mov [Buttons_debounce], a
    jmp main

;=====
;
; Task 1
;
;=====
;
; {
; do nothing

```

```

; }
;
;=====

Task1:
    jmp main

;=====
;
; Task 2
;
;=====
;
; {
; do nothing
; }
;
;=====

Task2:
    jmp main

;=====
;
; Task 3
;
;=====
;
; {
; do nothing
; }
;
;=====

Task3:
    jmp main

;=====
;
; Task 4
;
;=====
;
; {
; if (Buttons state has been stable for 5ms)
;     save Buttons
; }
;
;=====

Task4:
    iord Buttons_port
    cmp a, [Buttons_debounce]
    jnz main

    mov a, [Buttons_debounce]
    cpl a
    mov [EPl_FIFO], a
    mov a, ACKIN
    iowr EPl_MODE
    jmp main

;=====
;
; Task 5
;
;=====
;
; {
; }
;
;=====

Task5:
    jmp main

;=====

```

```

;
; Task 6
;
;=====
;
; {
; }
;
;=====

```

```

Task6:
    jmp main

```

```

;=====
;
; Task 7
;
;=====
;
; {
;     Write Ep2 info to LED's
;     And reset Mode of EP2
; }
;
;=====

```

```

Task7:
    mov a, [EP2_FIFO]
    iowr LEDs_port
    mov a, ACKOUT
    iowr EP2_MODE
    jmp main

```

```

include    "usbcode.asm"

```

```

IsrEp2:
    push a
    mov a, (EP0_IE+EP1_IE+EP2_IE)
    iowr EP_IE          ; enable EP interrupt here because no longer in GIER
    mov a, [interrupt_mask] ;
    iowr GIER          ;
    mov a, ACKOUT
    iowr EP2_MODE
    pop a
    reti

```

```

IsrSPI:
    push a
    mov a, (EP0_IE+EP1_IE)
    iowr EP_IE          ; enable EP interrupt here because no longer in GIER
    mov a, [interrupt_mask] ;
    iowr GIER          ;
    pop a
    reti

```

```

IsrCaptureA:
    push a
    mov a, (EP0_IE+EP1_IE)
    iowr EP_IE          ; enable EP interrupt here because no longer in GIER
    mov a, [interrupt_mask] ;
    iowr GIER          ;
    pop a
    reti

```

```

IsrCaptureB:
    push a
    mov a, (EP0_IE+EP1_IE)
    iowr EP_IE          ; enable EP interrupt here because no longer in GIER
    mov a, [interrupt_mask] ;
    iowr GIER          ;
    pop a
    reti

```

```

IsrWakeup:
    push a
    mov a, (EP0_IE+EP1_IE)
    iowr EP_IE          ; enable EP interrupt here because no longer in GIER
    mov a, [interrupt_mask] ;
    iowr GIER          ;

```



XPAGEON

```
=====
; Decode HID Class Request
=====
```

```
ClassRequest:
  cmp a, 20h
  jnz VendorRequest          ; if not a class request then check for a vendor request

  mov a, [bRequest]          ; check request is in range
  asl a                      ; multiply index by 2 for jump table
  cmp a, (end_HID_request_table - HID_request_table + 1)
  jnc SendStall

  jacc HID_request_table     ; jump to the routine that handles that request type

  db 00,00,00,00,00,00,00,00,00,00
```

XPAGEOFF

```
HID_request_table:
  jmp SendStall              ;reserved
  jmp GetReport
  jmp GetIdle
  jmp GetProtocol
  jmp SendStall              ;reserved
  jmp SendStall              ;reserved
  jmp SendStall              ;reserved
  jmp SendStall              ;reserved
  jmp SendStall              ;reserved
  jmp SetReport
  jmp SetIdle
  jmp SetProtocol
```

end\_HID\_request\_table:

XPAGEON

```
=====
; Decode Vendor Request
=====
; Frameworks does not support a vendor request; The user may insert
; code to do so here
```

```
VendorRequest:
  jmp SendStall              ;no vendor requests supported in Frameworks
```

```
=====
; Handle Request - Get Status
=====
; Device, Interface and Endpoint are all valid recipients for this request.
```

```
GetStatus:
  mov A, 2                  ; send two bytes
  mov [data_count], A

  mov a, [bmRequestType]   ;check recipient, and handle accordingly
  cmp a, DEVICE_TO_HOST
  jz GetDeviceStatus

  cmp a, ENDPOINT_TO_HOST
  jz GetEndpointStatus

  cmp a, INTERFACE_TO_HOST
  jnz SendStall            ;80, 81, 82 are only valid bmRequestTypes for this request
```

```
=====
; Recipient = Interface
=====
; The interface status is a 16-bit value (two bytes) that is always
; zero for both bytes.
```

```
GetInterfaceStatus:
  mov A, (get_interface_status_table - control_read_table)
  jmp SendRomData          ; send interface status to host
```

```
=====
; Recipient = Device
=====
; The device status is a 16-bit value (two bytes) with only D[1:0]
; defined. D0=0 specifies bus-powered, which never changes. D1
```

```

; reflects the status of the device_remote_wakeup feature. This
; feature can either be disabled (D1=0) or enabled (D1=1).

GetDeviceStatus:
    mov A, (get_dev_status_table - control_read_table)
;    add A, [remote_wakeup_status] ; get correct remote wakeup **** not yet supported
    jmp SendRomData ; send device status to host

;=====
; Recipient = Endpoint
;=====
; The endpoint status is a 16-bit value (two bytes) with only one
; bit (D0) defined. If D0=0, then the selected endpoint is not
; stalled. If D0=1, then the selected endpoint is stalled.

GetEndpointStatus:
    iord EP1_MODE
    and a, STALL_B
    jz .Send
    mov a, (stalled - get_endpoint_status_table) ;EP1 is stalled

.Send:
    add A, (get_endpoint_status_table - control_read_table)
    jmp SendRomData ; send endpoint status to host

;=====
; Handle Request - Clear Feature
;=====
; Endpoint and Device are supported recipients for this request. Interface is valid, but not supported.

ClearFeature:
    mov a, [bmRequestType]
    cmp a, HOST_TO_DEVICE
    jz ClearRemoteWakeup

    cmp a, HOST_TO_ENDPOINT
    jnz SendStall ;00, 02 are only valid bmRequestTypes for this request

;=====
; Recipient = Endpoint
;=====
; Clear the endpoint stall feature for the selected endpoint. This
; should also set the data 0/1 bit to Data0 if endpoint one is selected.

ClearEndpointStall:
    mov A, [wValue] ;
    cmp A, EP1_STALLED ; test for valid feature
    jnz SendStall ; stall unsupported features

    call NoDataControl ; handshake with host

    iord EP1_MODE
    and A, ~STALL_B ; clear stall and data 0/1 bits
    iowr EP1_MODE
    iord EP1_COUNT
    and a, ~DATA01_B
    iowr EP1_COUNT

    iord USB_SCR ; Set up EP1 to NAK INs
    and A, 0EFh ; INs are only ACKed when new data is
    iowr USB_SCR ; ready.

    ret

;=====
; Recipient = Device
;=====
; Remote Wakeup is the only Device feature

ClearRemoteWakeup:
    jmp SendStall ;**** not yet supported

    call NoDataControl ; handshake with host

    mov A, DISABLE_REMOTE_WAKEUP ; disable remote wakeup
    mov [remote_wakeup_status], A

    ret

```

```

;=====
;      Handle Request - Set Feature
;=====
; Endpoint and Device are supported recipients for this request. Interface is valid, but not supported.

SetFeature:
    mov a, [bmRequestType]
    cmp a, HOST_TO_DEVICE
    jz SetRemoteWakeup

    cmp a, HOST_TO_ENDPOINT
    jnz SendStall                ;00, 02 are only valid bmRequestTypes for this request

;=====
; Recipient = Endpoint
;=====
; Set the endpoint stall feature for the selected endpoint.

SetEndpointStall:
    mov A, [wValue]
    cmp A, EP1_STALLED          ; test for valid feature
    jnz SendStall              ; stall unsupported features

    call NoDataControl          ; handshake with host

    mov A, 80h                  ; stall endpoint one
    iowr EP1_MODE
    iord EP_IE                  ; and reenable endpoint 1
    or A, EP1_IE
    or A, EP2_IE
    iowr EP_IE
    ret

;=====
; Recipient = Device
;=====
; Remote Wakeup is the only Device feature
; If the device does not support remote wakeup then replace this code
; with <jmp SendStall>

SetRemoteWakeup:
    mov A, [wValue]
    cmp A, REMOTE_WAKEUP        ; test for valid feature
    jnz SendStall              ; stall unsupported features

    call NoDataControl          ; handshake with host

    mov A, ENABLE_REMOTE_WAKEUP ; enable remote internal wakeup variable
    mov [remote_wakeup_status], A
    ret

;=====
;      Handle Request - Set Address
;=====
; Device is the only valid recipient for this request.

SetAddress:
    mov a, [bmRequestType]
    cmp a, HOST_TO_DEVICE
    jnz SendStall

    mov a, [wValue]             ;check that the address is in range
    and a, 80h                 ;valid range is 0 to 7F
    jnz SendStall              ;

    call NoDataControl          ; handshake with host

.wait:
    iowr WDT
    iord EPO_MODE              ; Wait for the ACK.
    and A, 10h                 ; must wait for ACK before changing address
    jz .wait                   ; as the ACK will be sent to the old address.

    mov A, [wValue]            ; write new USB device address
    or A, 80h
    iowr USB_DA                ;
    ret

;=====

```

```

; Handle Request - Get Descriptor
;=====
; There are five standard descriptor types and 3 HID class descriptor types. The descriptor type is in
; the high byte of wValue. The descriptor index is in the low byte of wValue. The standard request to
; a device supports three of these types: device, configuration, and string.
; The standard request does not support interface or endpoint descriptor types

GetDescriptor:
    mov a, [bmRequestType]
    cmp a, DEVICE_TO_HOST
    jnz GetClassDescriptor          ; If recipient is not device, then check for Interface

    mov A, [wValueHi]
    asl a                          ; multiply index by 2 for jump table
    cmp a, (end_desc_type_table - desc_type_table + 1)
    jnc SendStall                  ; check that the descriptor type is valid

    jacc desc_type_table

GetClassDescriptor:
    mov a, [bmRequestType]
    cmp a, INTERFACE_TO_HOST
    jnz SendStall                  ; only valid recipient is Interface

    mov A, [wValueHi]              ; load descriptor type
    and a, dfh                     ; mask class bit
    asl a                          ; multiply index by 2 for jump table
    cmp a, (end_class_desc_type_table - class_desc_type_table + 1)
    jnc SendStall

    jacc class_desc_type_table

XPAGEOFF
desc_type_table:
    jmp SendStall                  ;no descriptor index 0
    jmp GetDeviceDescriptor
    jmp GetConfigurationDescriptor
    jmp GetStringDescriptor
end_desc_type_table:

class_desc_type_table:
    jmp SendStall                ; no descriptor index 0
    jmp GetHIDDescriptor
    jmp GetReportDescriptor
    jmp GetPhysicalDescriptor
end_class_desc_type_table:
XPAGEON

;=====
; Get Descriptor = Device
;=====

GetDeviceDescriptor:
    mov A, (end_device_desc_table - device_desc_table)
    mov [data_count], A          ; save the device descriptor length

    mov A, (device_desc_table - control_read_table)
    jmp SendRomData              ; pass a pointer to the device descriptor data

;=====
; Get Descriptor = Config
;=====
; When requested to send the config descriptor all of the Class, Interface,
; HID and Endpoint descriptors should be sent. For Draft 4 compliance, this
; order is mandatory

GetConfigurationDescriptor:
    mov A, (end_config_desc_table - config_desc_table)
    mov [data_count], A          ; save the config descriptor length

    mov A, (config_desc_table - control_read_table)
    jmp SendRomData              ; pass a pointer to the config descriptor data

;=====
; Get Descriptor = String
;=====

GetStringDescriptor:
    mov A, [wValue]              ; get string descriptor index

```

```

    cmp a, (end_string_length_table - string_length_table + 1)
    jnc SendStall                ; check index is in range

    index string_length_table    ; get the descriptor length
    mov [data_count], A        ;

    mov A, [wValue]             ; get string descriptor index
    index string_offset_table    ; get a pointer to the string descriptor data
    jmp SendRomData            ;

XPAGEOFF
string_length_table:
    db (USBStringDescription1 - USBStringLanguageDescription)
    db (USBStringDescription2 - USBStringDescription1)
    db (USBStringDescription3 - USBStringDescription2)
    db (USBStringDescription4 - USBStringDescription3)
    db (USBStringDescription5 - USBStringDescription4)
    db (USBStringEnd - USBStringDescription5)
end_string_length_table:

string_offset_table:
    db (USBStringLanguageDescription - control_read_table)
    db (USBStringDescription1 - control_read_table)
    db (USBStringDescription2 - control_read_table)
    db (USBStringDescription3 - control_read_table)
    db (USBStringDescription4 - control_read_table)
    db (USBStringDescription5 - control_read_table)
end_string_offset_table:
XPAGEON

;=====
; Get Descriptor = Report
;=====

GetReportDescriptor:
    mov A, (end_hid_report_desc_table - hid_report_desc_table)
    mov [data_count], A        ; save descriptor length

    mov A, (hid_report_desc_table - control_read_table)
    jmp SendRomData            ; send descriptor to host

;=====
; Get Descriptor = HID
;=====

GetHIDDescriptor:
    mov A, (Endpoint_Descriptor1 - Class_Descriptor)
    mov [data_count], A        ; save descriptor length

    mov A, ( Class_Descriptor - control_read_table)
    jmp SendRomData            ; send descriptor to host

;=====
; Get Descriptor = Physical
;=====
; Physical descriptors are not supported by frameworks. Users wishing
; to support them should add code to send them here.

GetPhysicalDescriptor:
    jmp SendStall                ;not supported by frameworks

;=====
; Handle Request - Set Descriptor
;=====
; Set Descriptor is not supported by frameworks. Users wishing to support this request should add code to
; do so here.

SetDescriptor:
    jmp SendStall                ;not supported by frameworks

;=====
; Handle Request - Get Configuration
;=====
; The only recipient supported for this request is Device
;
; Returns the current device configuration. Valid values are zero (unconfigured) and one (configured).

GetConfiguration:

```

```

mov a, [bmRequestType]
cmp a, DEVICE_TO_HOST
jnz SendStall

    mov A, 1                ; send one byte
    mov [data_count], A

    mov A, (get_configuration_status_table - control_read_table)
    add A, [configuration_status]    ; get correct configuration

    jmp SendRomData        ; send configuration to host

;=====
;      Handle Request - Get Configuration
;=====
; The only recipient supported for this request is Device
;
; Sets the configuration of the device to either unconfigured (0) or configured (1) based on wValue in the
; SETUP packet. USB spec rev 1.1 (para 9.1.1.5), requires Set Configuration to also clear the endpoint stall
; condition and re-initialize endpoints using data 0/1 toggle to Data0.

SetConfiguration:
    mov a, [bmRequestType]
    and a, HOST_TO_DEVICE        ; faster than <cmp a, 00h>
    jnz SendStall                ; stall if recipient not Device

    call NoDataControl           ;handshake with host

    mov A, [wValue]
    mov [configuration_status], A    ; store configuration byte

    iord EP1_COUNT                ; clear data 0/1 bit
    and A, ~DATA01_B
    iowr EP1_COUNT
    iord EP1_MODE
    and A, ~STALL_B
    iowr EP1_MODE

    mov A, [configuration_status]
    and a, ffh                    ; faster than <cmp a, 00h>
    jnz ConfigureDevice

UnconfigureDevice:
    iord EP_IE
    and A, ~EP1_IE                ; disable endpoint 1
    iowr EP1_IE

    mov A, [interrupt_mask]        ; disable endpoint one interrupts
    and A, EFh
    mov [interrupt_mask], A

    ret

ConfigureDevice:
    iord EP1_MODE                ; NAK IN packets until data is
    mov a, 0Eh
    iowr EP1_MODE
    iord EP_IE
    or a, EP1_IE
    iowr EP_IE                    ; enable endpoint 1

    mov A, [interrupt_mask]        ; enable gpio, ep1, 1ms 128us interrupts
    or A, ENUMERATED_MASK
    mov [interrupt_mask], A

    iord USB_SCR                ; NAK IN packets until data is
    and A, 0EFh                ; ready on endpoint one
    iowr USB_SCR

    ret

;=====
;      Handle Request - Get Interface
;=====
; The only recipient supported for this request is Interface
;
; The interface is an 8 bit value that is always zero since there are no alternate settings defined

GetInterface:

```

```

mov a, [bmRequestType]
cmp a, INTERFACE_TO_HOST
jnz SendStall

mov A, [wIndex]                ;which interface ?
  cmp A, 0                      ;only interface 0 defined
  jnz SendStall

  mov A, 1                      ;send one byte
  mov [data_count], A

  mov A, (get_interface_table - control_read_table)
  jmp SendRomData              ;send alternate interface to host

;=====
;      Handle Request - Set Interface
;=====
; The only recipient supported for this request is Interface
;
; The host may attempt to set the interface to an alternative setting. The frameworks does not support any
; alternate setting, but user firmware may so we do not stall that request, though it would be legal to do so.

SetInterface:
  mov a, [bmRequestType]
  cmp a, DEVICE_TO_HOST
  jnz SendStall

  mov a,[wIndex]                ; which interface ?
  cmp a,0                      ; only interface 0 defined
  jnz SendStall

  mov a,[wValue]                ; which setting
  cmp a, 0                      ; only setting 0 supported
  jnz SendStall

  jmp NoDataControl            ; ack the request

;=====
;      Handle Request - Get Report
;=====
; This is a HID Class request. The only recipient supported is Interface
;
; Get Report allows the host to receive a report via the control pipe. The report type is specified in
; the wValue high byte while the low byte has a report ID.

GetReport:
  mov a, [bmRequestType]
  cmp a, CLASS + INTERFACE_TO_HOST
  jnz SendStall

; Get Report support coming soon to a theater near you....

  jmp SendStall

;=====
;      Handle Request - Get Idle
;=====
; This is a HID Class request. The only recipient supported is Interface
;
; Microsoft inform us that this request is not used. As it is optional, it is stalled.

GetIdle:
  jmp SendStall

;=====
;      Handle Request - Get Protocol
;=====
; This is a HID Class request. The only recipient supported is Interface
;
; Get Protocol sends the current protocol status back to the host. For boot protocol, wValue=0.
; For report protocol, wValue=1.

GetProtocol:
  mov a, [bmRequestType]
  cmp a, CLASS + INTERFACE_TO_HOST
  jnz SendStall

  mov A, 1                      ; send one byte
  mov [data_count], A

```

```

    mov A, (get_protocol_status_table - control_read_table)
    add A, [protocol_status]          ; get correct configuration
    jmp SendRomData

;=====
;      Handle Request - Get Report
;=====
; This is a HID Class request. The only recipient supported is Interface
;
; Set Report allows the host to transmit a report via the control pipe. The report type is specified in
; the wValue high byte while the low byte has a report ID.

SetReport:
    mov a, [bmRequestType]
    cmp a, CLASS + HOST_TO_INTERFACE
    jnz SendStall

; Set Report support coming soon to a theater near you....

    jmp SendStall

;=====
;      Handle Request - Snet Idle
;=====
; This is a HID Class request. The only recipient supported is Interface
;
; Set Idle silences a particular report on the interrupt pipe until a new event occurs or the specified
; amount of time (wValue) passes.
;
; Microsoft inform us that this request is not used. As it is optional, it is stalled.

SetIdle:
    jmp SendStall

;=====
;      Handle Request - Get Protocol
;=====
; This is a HID Class request. The only recipient supported is Interface
;
; Set Protocol switches between the boot protocol and the report protocol. For boot protocol, wValue=0.
; For report protocol, wValue=1.

SetProtocol:
    mov a, [bmRequestType]
    cmp a, CLASS + HOST_TO_INTERFACE
    jnz SendStall

    mov A, [wValue]
    mov [protocol_status], A          ; write new protocol value

    jmp NoDataControl                ; handshake with host

;=====
;*****USB library main routines *****
;=====

;=====
; Function - Send ROM Data
;=====
; This function controls the sending of data from the descriptor table
; starting from the table index passed in the accumulator. The number
; of bytes to be sent is passed using the global variable <data_count>

SendRomData:
    mov [data_start], A              ; save start index

    call GetDescriptorLength          ; correct the descriptor length
    call ControlRead                  ; send the data

    ret

;=====
; Function Send Stall
;=====
; Send a stall to indicate the requested function is not supported

SendStall:
    ;mov A, 0                          ;

```



```

mov A, [data_count]
cmp A, 00h
jz DmaLoadDone

DmaLoadLoop:          ; loop to load data into the data buffer
mov A, [data_start]
index control_read_table
mov [X + EP0_FIFO], A      ; load dma buffer

inc [data_start]
inc X
inc [loop_counter]

dec [data_count]          ; exit if descriptor
jz DmaLoadDone           ; is done

mov A, [loop_counter]     ; or 8 bytes sent
cmp A, 08h
jnz DmaLoadLoop

DmaLoadDone:
iord EP0_COUNT
iord EP0_MODE             ; check setup bit
and A, 80h                ; if not cleared, another setup
jnz ControlReadStatusStage ; has arrived so exit

mov A, [endp0_data_toggle]
xor A, DATA01_B
mov [endp0_data_toggle], A

;or A, 80h
or A, [loop_counter]
iowr EP0_COUNT

ControlReadStatusStage:  ; OUT at end of data transfer
pop X                    ; restore X from stack
ret

;=====
; ROM Tables
;=====
;

XPAGEOFF

control_read_table:
device_desc_table:
db (end_device_desc_table - device_desc_table) ; Descriptor length (18 bytes)
db 01h ; Descriptor type (Device)
db 10h,01h ; Complies with USB Spec. Release (1.1)
db 00h ; Class code (0)
db 00h ; Subclass code (0)
db 00h ; Protocol (No specific protocol)
db 08h ; Max. packet size for EP0 (8 bytes)
db 25h,09h ; Vendor ID (Cypress) - User must change this to own Vid
db 36h,12h ; Product ID (Frameworks)- User must Change this
db 00h,00h ; Device release number
db 01h ; Mfr string descriptor index
db 02h ; Product string descriptor index
db 00h ; Serial Number string descriptor index (None)
db 01h ; Number of possible configurations (1)
end_device_desc_table:

config_desc_table:
db (Interface_Descriptor - config_desc_table) ; Descriptor length (9 bytes)
db 02h ; Descriptor type (Configuration)
db (end_config_desc_table - config_desc_table) ,00h
; Total length of config, i/f, HID, and EP descriptors
db 01h ; Interface supported (1)
db 01h ; Configuration value (1)
db 04h ; Index of string descriptor
db 80h ; Configuration (Bus powered)
db 10h ; Max power consumption (32mA) MUST be < 100mA if bus powered

Interface_Descriptor:
db (Class_Descriptor - Interface_Descriptor) ; Descriptor length (9 bytes)
db 04h ; Descriptor type (Interface)
db 00h ; Number of interface (0)
db 00h ; Alternate setting (0)

```

```

db 02h          ; Number of interface endpoint (1)
db 03h          ; Class code ( )
db 00h          ; Subclass code ( )
db 00h          ; Protocol code ( )
db 05h          ; Index of string( )

```

Class\_Descriptor:

```

db (Endpoint_Descriptor1 - Class_Descriptor) ; Descriptor length (9 bytes)
  db 21h          ; Descriptor type (HID)
db 00h,01h      ; HID class release number (1.00)
db 00h          ; Localized country code (None)
db 01h          ; # of HID class dscrptr to follow (1)
db 22h          ; Report descriptor type (HID)
dwl (end_hid_report_desc_table - hid_report_desc_table) ; Total length of report descriptor (2 bytes)

```

Endpoint\_Descriptor1:

```

db (Endpoint_Descriptor2 - Endpoint_Descriptor1) ; Descriptor length (7 bytes)
db 05h          ; Descriptor type (Endpoint)
db 81h          ; Encoded address (Respond to IN, 1 endpoint)
db 03h          ; Endpoint attribute (Interrupt transfer)
db 01h,00h      ; Maximum packet size (1 bytes)
db 0Ah          ; Polling interval (10 ms)

```

Endpoint\_Descriptor2:

```

db (end_config_desc_table - Endpoint_Descriptor2) ; Descriptor length (7 bytes)
db 05h          ; Descriptor type (Endpoint)
db 02h          ; Encoded address (Respond to OUT, 2 endpoint)
db 03h          ; Endpoint attribute (Interrupt transfer)
db 01h,00h      ; Maximum packet size (1 bytes)
db 0Ah          ; Polling interval (10 ms)

```

end\_config\_desc\_table:

; This report descriptor is taken from Appendix A12 of the HID Usage Tables

hid\_report\_desc\_table:

```

db 06h, A0h, FFh ; Usage Page (vendor defined) FFA0
db 09h, 01h ; Usage (vendor defined)
db A1h, 01h ; Collection (Application)
db 09h, 02h ; Usage (vendor defined)
db A1h, 00h ; Collection (Physical)
db 06h, A1h, FFh ; Usage Page (vendor defined)

```

;The input report

```

db 09h, 03h ; usage - vendor defined
db 09h, 04h ; usage - vendor defined
db 15h, 80h ; Logical Minimum (-128)
db 25h, 7Fh ; Logical Maximum (127)
db 35h, 00h ; Physical Minimum (0)
db 45h, FFh ; Physical Maximum (255)
; db 66h, 00h, 00h; Unit (None (2 bytes))
db 75h, 08h ; Report Size (8) (bits)
db 95h, 01h ; Report Count (1) (fields)
db 81h, 02h ; Input (Data, Variable, Absolute)

```

;The output report

```

db 09h, 05h ; usage - vendor defined
db 09h, 06h ; usage - vendor defined
db 15h, 80h ; Logical Minimum (-128)
db 25h, 7Fh ; Logical Maximum (127)
db 35h, 00h ; Physical Minimum (0)
db 45h, FFh ; Physical Maximum (255)
; db 66h, 00h, 00h; Unit (None (2 bytes))
db 75h, 08h ; Report Size (8) (bits)
db 95h, 01h ; Report Count (1) (fields)
db 91h, 02h ; Output (Data, Variable, Absolute)

```

```

db C0h ; End Collection

```

```

db C0h ; End Collection

```

end\_hid\_report\_desc\_table:

```

;=====
; These tables are the mechanism used to return status information to the
; host. The status can be either device, interface, or endpoint.

```

get\_dev\_status\_table:

```

db 00h, 00h ; remote wakeup disabled, bus powered

```

wakeup\_enabled:

```

    db    02h, 00h        ; remote wakeup enabled, bus powered

get_interface_status_table:
    db    00h, 00h        ; always return both bytes zero

get_endpoint_status_table:
    db    00h, 00h        ; not stalled
stalled:
    db    01h, 00h        ; stalled

get_configuration_status_table:
    db    00h            ; not configured
    db    01h            ; configured

get_protocol_status_table:
    db    00h            ; boot protocol
    db    01h            ; report protocol

get_interface_table:
    db    00h            ; no alternate setting

;=====
; String Descriptors      Do not make the string descriptor too long
;                          because the entire rom lookup table cannot be
;                          longer than 256 bytes

; string 0
USBStringLanguageDescription:
    db (USBStringDescription1 - USBStringLanguageDescription)
    db 03h                ; Type (3=string)
    db 09h                ; Language: English
    db 04h                ; Sub-language: Default

; string 1
USBStringDescription1: ; IManufacturerName
    db (USBStringDescription2 - USBStringDescription1)
    db 03h                ; Type (3=string)
    dsu "CYPRESS" ;

; string 2
USBStringDescription2: ; IProduct
    db (USBStringDescription3 - USBStringDescription2)
    db 03h                ; Type (3=string)
    dsu "CYPRESS FRAMEWORKS" ;

;string 3
USBStringDescription3: ; serial number
                        ; If a SN is used, this must be unique
                        ; for every device or the device may
                        ; not enumerate properly

; string 4
USBStringDescription4: ; configuration string descriptor
    db (USBStringDescription5 - USBStringDescription4)
    db 03h                ; Type (3=string)
    dsu "FRAMEWORKS" ;

;string 5
USBStringDescription5: ; interface string descriptor
    db (USBStringEnd - USBStringDescription5)
    db 03h                ; Type (3=string)
    dsu "EP1" ;

USBStringEnd:
XPAGEON

```