Digital Design: An Embedded Systems Approach Using VHDL

Chapter 8
I/O Interfacing

Portions of this work are from the book, Digital Design: An Embedded Systems Approach Using VHDL, by Peter J. Ashenden, published by Morgan Kaufmann Publishers, Copyright 2007 Elsevier Inc. All rights reserved.
I/O Devices and Transducers

- Transducers convert between real-world effects and digital representation
  - Input transducers: sensors
    - May require analog-to-digital converter (ADC)
  - Output transducers: actuators
    - May require digital-to-analog converter (DAC)

- Human-interface devices
  - Buttons, switches, knobs, keypads, mouse
  - Indicators, displays, speakers
Kaypads & Keyboards

- Recall switches and debouncing
- Keypad: array of push-button switches

Scan the matrix by driving each row low one at a time and see if any of the column lines become low.
Knobs & Position Encoders

- In analog circuits, use a variable resistor
- In digital circuits, could use pushbuttons
  - E.g., volume up/down
  - Not as easy to use as knobs or sliders
- Can use a position encoder attached to a knob
  - Recall Gray code encoder
Incremental Encoder

If absolute position is not important, incremental encoder is simpler.

- Counterclockwise:
  - A
  - B

- Clockwise:
  - A
  - B
Analog Inputs

- Physical effect produces an analog voltage or current
- Microphone
  - In phones, cameras, voice recorders, …
- Accelerometer
  - In airbag controllers
- Fluid-flow sensors
  - In industrial machines, coffee machines, …
- Gas detectors
  - In safety equipment
Analog-to-Digital Converters

- Basic element: analog comparator
- Flash ADC
  - Simple, fast, but uses many comparators
- Resolution
  - Number of output bits
Successive Approximation ADC

- Initial approximation: 01111111
  - Comparator output gives $d_7$
    - 1 if $V_{in}$ is higher than 01111111, 0 otherwise
- Next approximation: $d_70111111$
  - Comparator output gives $d_6$
  - Next approximation: $d_7d_6011111$, etc
LED Indicators

- Single LED shows 1-bit state
  - On/off, busy/ready, error/ok, …

- Brightness depends on current
  - Determined by resistor
  - \( I = \left( +V - V_{LED} - V_{OL} \right) / R \)
7-Segment LED Displays

- Each digit has common anodes or common cathodes
  - Scan: turn on one digit at a time
Example: Multiplexed Display

- Four BDC inputs, 10MHz clock
  - Turn on decimal point of leftmost digit only
  - 50Hz scan cycle (200Hz scan clock)

```vhdl
library ieee;
use ieee.std_logic_1164.all, ieee.numeric_std.all;

entity display_mux is
  port ( clk, reset : in std_logic;
         bcd0, bcd1,
         bcd2, bcd3 : in unsigned(3 downto 0);
         anode_n    : out std_logic_vector(3 downto 0);
         segment_n  : out std_logic_vector(7 downto 0) );
end entity display_mux;
```
Example: Multiplexed Display

```vhdl
architecture rtl of display_mux is

constant clk_freq   : natural := 10000000;
constant scan_clk_freq : natural := 200;
constant clk_divisor   : natural := clk_freq
                            / scan_clk_freq;

signal scan_clk  : std_logic;
signal digit_sel : unsigned(1 downto 0);
signal bcd       : unsigned(3 downto 0);
signal segment   : std_logic_vector(7 downto 0);

begin
```
Example: Multiplexed Display

-- Divide master clock to get scan clock

scan_clk_gen : process (clk) is
    variable count : natural range 0 to clk_divisor - 1;
begin
    if rising_edge(clk) then
        if reset = '1' then
            count := 0;
            scan_clk <= '0';
        elsif count = clk_divisor - 1 then
            count := 0;
            scan_clk <= '1';
        else
            count := count + 1;
            scan_clk <= '0';
        end if;
    end if;
end process scan_clk_gen;
Example: Multiplexed Display

```
-- increment digit counter once per scan clock cycle

digit_counter : process (clk) is
begin
  if rising_edge(clk) then
    if reset = '1' then
      digit_sel <= "00";
    elsif scan_clk = '1' then
      digit_sel <= digit_sel + 1;
    end if;
  end if;
end process digit_counter;
```
Example: Multiplexed Display

```
-- multiplexer to select a BCD digit

with digit_sel select
  bcd <= bcd0 when "00",
        bcd1 when "01",
        bcd2 when "10",
        bcd3 when others;

-- activate selected digit's anode

with digit_sel select
  anode_n <= "1110" when "00",
            "1101" when "01",
            "1011" when "10",
            "0111" when others;
```

-- Selected Signal Assignments
Example: Multiplexed Display

-- 7-segment decoder for selected digit

with bcd select
    segment(6 downto 0) <= "0111111" when "0000", -- 0
                      "0000110" when "0001", -- 1
                      "1011011" when "0010", -- 2
                      "1001111" when "0011", -- 3
                      "1100110" when "0100", -- 4
                      "1101101" when "0101", -- 5
                      "1111101" when "0110", -- 6
                      "0000111" when "0111", -- 7
                      "1111111" when "1000", -- 8
                      "1101111" when "1001", -- 9
                      "1000000" when others; -- "-"

-- decimal point is only active for digit 3
segment(7) <= '1' when digit_sel = "11" else '0';

-- segment outputs are negative logic
segment_n <= not segment;

end architecture rtl;
Liquid Crystal Displays (LCDs)

- **Advantages**
  - Low power
  - Readable in bright ambient light conditions
  - Custom segment shapes

- **Disadvantages**
  - Require backlight for dark conditions
  - Not as robust as LEDs

- **LCD panels**
  - Rectangular array of pixels
  - Can be used for alphanumerical/graphical display
  - Controlled by a small microcontroller
Actuators & Valves

- Actuators cause a mechanical effect
- Solenoid: current in coil moves armature
  - Can attach rods, levers, etc
to translate the movement
- Solenoid valve
  - Armature controls fluid
  or gas valve
- Relay
  - Armature controls
electrical contacts

The magnetic field created by the current pulls the bar inward.
Motors

- Can provide angular position or speed
  - Use gears, screws, etc to convert to linear position or speed
- Stepper motors
  - Rotate in discrete steps
Motors

- Servo-motors
  - DC motor, speed controlled by varying the drive voltage
  - Use feedback to control the speed or to drive to a desired position
  - Requires a position sensor or tachometer

- Servo-controller
  - A digital circuit or an embedded processor
  - Compensates for non-ideal mechanical effects
R-string DAC
- Voltage divider and analog multiplexer
- Requires $2^n$ precision resistors
- **R-2R ladder DAC**
  - Sums binary-weighted currents
  - Requires $2^n$ matched resistors
I/O Controllers

- An embedded processor needs to access input/output data
- I/O controller
  - Circuit that connects I/O device to a processor
  - Includes control circuits
  - Input registers: for reading data
  - Output registers: for writing data
- I/O ports
  - Registers accessible to embedded software
Simple I/O Controller

- Just contains input and/or output registers
- Select among them using a port address

```vhdl
entity gumnut is
  port ( clk_i : in std_logic;
         rst_i : in std_logic;
         ...:
         port_cyc_o : out std_logic;
         port_stb_o : out std_logic;
         port_we_o : out std_logic;
         port_ack_i : in std_logic;
         port_adr_o : out unsigned(7 downto 0);
         port_dat_o : out std_logic_vector(7 downto 0);
         port_dat_i : in std_logic_vector(7 downto 0);
         ... );
end entity gumnut;
```
Example: Keypad Controller

- Output register for row drivers
- Input register for column sensing
Example: Keypad Controller

```vhdl
library ieee; use ieee.std_logic_1164.all;

entity keypad_controller is
  port ( clk_i    : in std_logic;
        cyc_i    : in std_logic;
        stb_i    : in std_logic;
        we_i     : in std_logic;
        ack_o    : out std_logic;
        dat_i    : in std_logic_vector(7 downto 0);
        dat_o    : out std_logic_vector(7 downto 0);
        keypad_row : out std_logic_vector(3 downto 0);
        keypad_col : in std_logic_vector(2 downto 0) );
end entity keypad_controller;

architecture rtl of keypad_controller is
  signal col_synch : std_logic_vector(2 downto 0);
begin
  ...
```

Example: Keypad Controller

```vhdl
... row_reg : process (clk_i) is
begin
  if rising_edge(clk_i) then
    if cyc_i = '1' and stb_i = '1' and we_i = '1' then
      keypad_row <= dat_i(3 downto 0);
    end if;
  end if;
end process row_reg;

col_synchronizer : process (clk_i) is
begin
  if rising_edge(clk_i) then
    dat_o <= "00000" & col_synch;
    col_synch <= keypad_col;
  end if;
end process col_synchronizer;

ack_o <= cyc_i and stb_i;
end architecture rtl;
```

Remember that the keypad signals are asynchronous and need to be synchronized.
Control/Status Registers

- Control register
  - Contains bits that govern operation of the I/O device
  - Written by processor

- Status register
  - Contains bits that reflect status of device
  - Read by processor

- Either or both may be needed in an input or output controller
Example: ADC Controller

- Successive approximation ADC
  - 1 × analog input with sample/hold
  - 4 × analog reference voltages

- Control register
  - Selects reference voltage
  - Hold input voltage & start ADC

- Status register
  - Is conversion done?

- Input data register
  - Converted data
Example: ADC Controller

VHDL

port_ack_o

port_cyc_i

port_stb_i

D0
Q0
Y1

D0
Y1

Y2

Y3

port_dat_i

port_cyc_i

port_stb_i

port_we_i

rst_i

clk_i

port_adr_i(0)

Vin

Vf_0

Vf_1

Vf_2

Vf_3

Digital Design — Chapter 8 — I/O Interfacing

Ashenden Designs
Autonomous I/O Controllers

- Independently sequence operation of a device
  - Processor initiates actions
  - Controller notifies processor of events, such as data availability, error condition, ...

- Processor can perform other operations concurrently

- Device operation not limited by processor performance or load
Example: LCD Module

- Rectangular array of pixels
  - Row and column connections
  - Controller scans rows, activates columns
- Image or character data stored in a small memory in the controller
  - Updated by an attached processor
Direct Memory Access (DMA)

- For high-speed input or output
  - Processor writes starting address to a control register
  - Controller transfers data to/from memory autonomously
  - Notifies processor on completion/error
- Reduces load on processor
- Common with accelerators
Parallel Buses

- Interconnect components in a system
  - Transfer bits of data in parallel
- Conceptual structure
  - All inputs and output connected
- In practice
  - Can’t tie multiple outputs together
Multiplexed Buses

- Use multiplexer(s) to select among data sources
  - Can partition to aid placement on chip
Example: Wishbone Bus

- Non-proprietary bus spec
  - OpenCores Organization

- Gumnut uses simple form of Wishbone
  - One bus for each of instruction memory, data memory, and I/O ports
  - “…_o” denotes output
  - “…_i” denotes input
Example: Wishbone Bus

VHDL

Gumnut
- port_adr_o
- port_dat_i
- port_ack_i
- port_cyc_o
- port_we_o
- port_stb_o

Keypad Controller
- dat_i
- cyc_i
- we_i
- stb_i

ADC Controller
- adr_i(0)
- dat_i
- cyc_i
- we_i
- stb_i

Digital Design — Chapter 8 — I/O Interfacing

Ashenden Designs
Tristate Buses

- Use tristate drivers for data sources
  - Can “turn-off” (Hi-Z) when not supplying data
- Simplified bus wiring

\[ \text{bus}(0) \]
\[ \text{bus}(1) \]
\[ \text{bus}(2) \]
\[ \vdots \]
\[ \text{bus}(n) \]

\[ \text{d}(0) \]
\[ \text{d}(1) \]
\[ \text{d}(2) \]
\[ \vdots \]
\[ \text{d}(n) \]

\[ \text{en} \]
Tristate Bus Issues

- Floating bus can cause spurious switching
  - Use pull-up resistors or weak keepers
- Need to avoid driver contention
  - Dead cycle between turn-off and turn-on
  - Or delayed enable

- Not all CAD tools and implementation fabrics support tristate buses
Tristate Drivers in VHDL

- Assign 'Z' to a signal to turn driver off

- Example: single-bit driver
  ```vhdl
  d_out <= d_in when d_en = '1'
           else 'Z';
  ```

- Example: multi-bit driver
  ```vhdl
  bus_o <= dat when dat_en = '1'
           else "ZZZZZZZZZ";
  ```

- Any other driver contributing '0' or '1' overrides 'Z' value

- Std_logic signals are resolved
Altera I/O : ALT_IOBUF Primitive

The ALT_IOBUF primitive allows you to do the following:

- Make a location assignment
- Make an I/O standard assignment
- Make a drive strength (current strength) assignment
- Make a slow slew rate assignment
- Enable bus-hold circuitry
- Enable a weak pull-up resistor
- Make an on-chip termination (OCT) assignment to a bidirectional pin from a lower-level entity
- This primitive is available for supported device (Cyclone III and Stratix III) families.
component alt_iobuf
    generic(
        io_standard : string := "NONE";
        current_strength : string := "NONE";
        slew_rate : integer := -1;
        slow_slew_rate : string := "NONE";
        location : string := "NONE";
        enable_bus_hold : string := "NONE";
        weak_pull_up_resistor : string := "NONE";
        termination : string := "NONE";
        input_termination : string := "NONE";
        output_termination : string := "NONE" );
    port(
        i : in std_logic;
        oe : in std_logic;
        io : inout std_logic;
        o : out std_logic);
end component;

LIBRARY altera;
USE altera.altera_primitives_components.all;
Example: SN74x16541

```vhdl
library ieee; use ieee.std_logic_1164.all;

entity sn74x16541 is
    port
        ( en1_1, en1_2, en2_1, en2_2 : in std_logic;
            a1, a2 : in std_logic_vector(7 downto 0);
            y1, y2 : out std_logic_vector(7 downto 0) );
end entity sn74x16541;

architecture rtl of sn74x16541 is
begin
    y1 <= a1 when en1_1 = '0' and en1_2 = '0' else (others => 'Z');
    y2 <= a2 when en2_1 = '0' and en2_2 = '0' else (others => 'Z');
end architecture rtl;
```
Unknown Values in VHDL

- What if two drivers are turned on?
  - One driving '0', the other driving '1'
  - Resolved value is 'X' — unknown
  - Can test for 'X' during simulation

- 'Z' and 'X' are not electrical logic levels
  - Notations for simulation and synthesis
  - Real logic levels are only 0 or 1
Open-Drain Buses

- Bus is 0 if any driver pulls it low.
- If all drivers are off, bus is pulled high.
  - Wired-AND
- Can also use open-collector drivers.
Open-Drain Drivers in VHDL

- Assign '0' or 'Z' to model driver
- Model pull-up using 'H'
  - \texttt{bus\_sig} \Leftarrow \texttt{'H'};
  - Weak high, overridden by '0'
- Logic operations
  - \texttt{gated\_sig} \Leftarrow \texttt{bus\_sig and sig\_en};
- Strip strength for comparisons
  - \texttt{mux\_out} \Leftarrow
    \begin{align*}
    \texttt{a1 when To\_X01(bus\_sig) = '1'} \quad \text{else} \\
    \texttt{a0 when To\_X01(bus\_sig) = '0'} \quad \text{else} \\
    "XXXXXXX"
    \end{align*}
Bus Protocols

- Specification of signals, timing, and sequencing of bus operations
  - Allows independent design of components
  - Ensures interoperability

- Standard bus protocols
  - PCI, VXI, …
    - For connecting boards in a system
  - AMBA (ARM), CoreConnect (IBM), Wishbone (Open Cores)
    - For connecting blocks within a chip
Example: Gumnut Wishbone

- Minimal 8-bit subset used for I/O ports

- Signals
  - port_cyc_o: “cycle” control for sequence of port operations
  - port_stb_o: “strobe” control for an operation
  - port_we_o: write enable
  - port_ack_i: acknowledge from addressed port
  - port_adr_o: 8-bit port address
  - port_dat_o: 8-bit data output from Gumnut
  - port_dat_i: 8-bit data input to Gumnut
Gumnut Wishbone Write

<table>
<thead>
<tr>
<th>clk</th>
<th>port_adr_o</th>
<th>port_dat_o</th>
<th>port_cyc_o</th>
<th>port_stb_o</th>
<th>port_we_o</th>
<th>port_ack_i</th>
</tr>
</thead>
</table>

No wait cycles

One wait cycle
VHDL

Gumnut Wishbone Read

clk
port_adr_o
port_cyc_o
port_stb_o
port_we_o
port_dat_i
port_ack_i

No wait cycles

One wait cycle
Serial Transmission

- Bits transmitted one after another on a single wire
  - Can afford to optimize the wire for speed
- C.f. parallel transmission, one wire per bit
  - Requires more wires
    - Cost per wire, greater area for wiring, complexity of place & route
  - Requires more pins
    - Cost of larger package
- Other effects
  - Crosstalk, skew, delay due to increased area
- Serializer/deserializer (serdes)
  - Converts between parallel and serial form
Example: 64-bit Serdes

- Bit order is arbitrary, provided both ends agree
  - Often specified by standards
NRZ Transmission

- Non-Return to Zero
  - Just set signal to high or low for each bit time
  - No indication of boundary between bit times
  - Need to synchronize transmitter and receiver separately
    - E.g., by a common clock and control signals, as in previous example
Start/Stop Bit Synchronization

- Hold signal high when there is no data
- To transmit
  - Drive signal low for one bit time (start bit)
  - Then drive successive data bits
  - Then drive signal high for one bit time (stop bit)

```
   1 1 1 0 0 1 0 0
```

1 1 1 0 0 1 0 0
Universal Asynchronous Receiver/Transmitter

- Common I/O controller for serial transmission using NRZ with start/stop bits
- Relies on Tx and Rx clocks being approximately the same frequency
Manchester Encoding

- Combine Tx clock with Tx data
  - Ensures regular edges in the serial signal
- Example: Manchester encoding
  - Transition in the middle of each bit time
    - 0: low-to-high transition
    - 1: high-to-low transition
    - May need a transition at the start of a bit time

\[\begin{align*}
1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\
\text{Tx_clk} & & & & & & & \\
\text{Tx_D} & & & & & & & 
\end{align*}\]
Clock Recovery

- Transmitter sends preamble before data
  - A sequence of encoded 1 bits
  - Serial signal then matches Tx clock
- Receiver uses a phase-locked loop (PLL) to match Rx clock to Tx clock

\begin{figure}
\centering
\begin{tikzpicture}
\draw[thick] (0,0) -- (15,0) node[below] {locked};
\draw[thick] (0,2) -- (15,2) node[below] {Tx_clk} coordinate (A);
\draw[thick] (0,3) -- (15,3) node[below] {Tx_D} coordinate (B);
\draw[thick] (0,4) -- (15,4) node[below] {Rx_clk} coordinate (C);
\draw[thick] (0,5) -- (15,5) node[below] {idle} coordinate (D);
\draw[thick] (0,6) -- (15,6) node[below] {data word} coordinate (E);
\end{tikzpicture}
\end{figure}
Serial Interface Standards

- Connection of I/O devices to computers
- Connection of computers in networks
- Use of standards reduces design effort
  - Reuse off-the-shelf components or IP
- RS-232: NRZ, start/stop bits
  - Originally for modems, now widely used for low-bandwidth I/O
Serial Interface Standards

- **I²C: Inter-Integrated Circuit bus**
  - 2 wires (NRZ data, clock), open drain
  - Simple protocol, low cost, 10kb/s–3.4Mb/s

- **USB: Universal Serial Bus**
  - For connecting I/O devices to computers
  - Differential signaling on 2 wires
  - 1.5Mb/s, 12Mb/s, 480Mb/s, …, complex protocol
  - IP blocks available

- **FireWire: IEEE Std 1394**
  - 2 differential pairs (data, synch)
  - 400Mb/s, 3.2Gb/s, complex protocol
I²C Example: Temperature Sensor

- Gumnut, Analog Devices AD7414
- I²C controller IP from OpenCores repository
I/O Software

- Use input and output instructions to access I/O controller registers
- I/O devices interact with the physical world
  - Software must respond to events when they occur
  - It must be able schedule activity at specific times or at regular intervals
  - *Real-time* behavior
Polling

- Software repeatedly reads I/O status to see if an event has occurred
  - If so, it performs the required action
- Multiple controllers
  - Software executes a polling loop, checking controllers in turn
- Advantage: simple I/O controllers
- Disadvantages
  - Processor is continually busy, consuming power
  - Delay in dealing with an event if processor is busy with another event
Polling Example

- Safety monitor in factory automation
  - Gumnut core
  - 16 alarm inputs
    - One per bit in registers at addresses 16 & 17
    - 0 ⇒ ok, 1 ⇒ abnormal condition
  - Temp sensor ADC at address 20
    - 8-bit binary code for °C
    - Above 50°C is abnormal
  - Alarm output at address 40
    - 0 ⇒ ok, 1 ⇒ ring alarm bell
Polling Example

```
alarm_in_1: equ 16      ; address of alarm_in_1 input register
alarm_in_2: equ 17      ; address of alarm_in_2 input register
temp_in: equ 20        ; address of temp_in input register
alarm_out: equ 40      ; address of alarm_out output register
max_temp: equ 50       ; maximum permissible temperature

poll_loop:  inp r1, alarm_in_1
            sub r0, r1, 0
            bnz set_alarm ; one or more alarm_in_1 bits set
            inp r1, alarm_in_2
            sub r0, r1, 0
            bnz set_alarm ; one or more alarm_in_2 bits set
            inp r1, temp_in
            sub r0, r1, max_temp
            bnc set_alarm ; temp_in > max_temp
            out r0, alarm_out ; clear alarm_out
            jmp poll_loop

set_alarm:  add r1, r0, 1
            out r1, alarm_out ; set alarm_out bit 1 to 1
            jmp poll_loop
```
Interrupts

- I/O controller notifies processor when an event occurs
  - Processor interrupts what it was doing
  - Executes interrupt service routine
    - A.k.a. interrupt handler
  - Then resumes interrupted task
  - May enter low-power standby

- Some systems prioritize interrupt requests
  - Allow higher priority events to interrupt service of lower priority events
Interrupt Mechanisms

- Interrupt request signal
- Means of disabling/enabling interrupts
  - So processor can execute *critical regions*
- Save processor state on an interrupt
  - So interrupted task can be resumed
- On interrupt, disable further interrupts
  - Until processor has saved state
- Find the handler code for the event
  - *Vector*: address of handler, or index into table of handler addresses
- Instruction to return from handler
  - Restoring saved state
Gumnut Interrupt Mechanisms

- `int_req` signal
- `disi` and `enai` instructions
- On interrupt, PC, Z, and C saved in special registers
- On interrupt, further interrupts are disabled
- Handler code starts at address 1
  - Gumnut sets PC to 1
- `reti` instruction
  - Restores PC, Z, and C from special registers, re-enables interrupts
Interrupt Acknowledgment

- Process may not respond immediately
  - But must tell controller when it does
  - Controller then deactivates request
    - To avoid multiple interrupts for one event
- Processor *acknowledges* the request
  - E.g., `int_ack` signal on Gumnut
  - Alternative: reading a status register
Example: Sensor Controller

- 8-bit input from sensor
- Interrupt request on change of value
**Example: Sensor Handler**

```vhdl
data

saved_r1: bss 1

.text

sensor_data: equ 0 ; address of sensor data

; input register

org 1

stm r1, saved_r1

inp r1, sensor_data

... ; process the data

ldm r1, saved_r1

reti
```
Timers

- Real-time clock (RTC)
  - Generates periodic interrupts
  - Uses a counter to divide system clock
  - Control register for divisor

- Interrupt handler can perform periodic tasks
  - E.g., activate next digit of a scanned display
Example: RTC for Gumnut

- 10µs timebase, divided by a down counter
  - Initial count loaded from a register
  - Interrupt triggered on count value = 0

<table>
<thead>
<tr>
<th>Offset</th>
<th>Output Registers</th>
<th>Input Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>start_count</td>
<td>count_value</td>
</tr>
<tr>
<td>1</td>
<td>E</td>
<td>0000000000000001</td>
</tr>
</tbody>
</table>

Interrupt Enable

Interrupt Triggered
Real-Time Executives

- Control program
  - A.k.a. real-time operating system (RTOS)
  - Timing based on a real-time clock
  - Schedules execution of tasks
    - In response to interrupts and timer events

- Can also manage other resources
  - Memory allocation
  - Storage (file system)
  - Use of I/O controllers
  - Use of accelerators
Example: Gumnut Executive

- RTC based at address 16
- Calls task_2ms every 2ms

```vhdl
;;;---------------------------------------------------------
;;; Program reset: jump to main program

text

org 0

jmp main

;;;---------------------------------------------------------
;;; Port addresses

rtc_start_count: equ 16 ; data output register
rtc_count_value: equ 16 ; data input register
rtc_int_enable: equ 17 ; control output register
rtc_int_status: equ 17 ; status input register
```
Example: Gumnut Executive

```vhdl
;;; -----------------------------
;;; init_interrupts: Initialize 2ms periodic interrupt, etc.

data
rtc_divisor:   equ     199     ; divide 100kHz down
               ; to 500Hz
rtc_int_flag:  bss     1

text
init_interrupts: add     r1, r0, rtc_divisor
                 out     r1, rtc_start_count
                 add     r1, r0, 1
                 out     r1, rtc_int_enable
                 stm     r0, rtc_int_flag
                 ...     ; other initializations
                 ret
```
Example: Gumnut Executive

```vhdl
;;; --- ---------------------------- Interrupt handler ----------------------------
;;; Interrupt handler

data
int_r1: bss 1 ; save location for
         ; handler registers

text
org 1

int_handler: stm r1, int_r1 ; save registers
check_rtc: inp r1, rtc_status ; check for
           ; RTC interrupt
           sub r0, r1, 0
           bz check_next
           add r1, r0, 1
           stm r1, rtc_int_flag ; tell main
           ; program

check_next:

int_end: ldm r1, int_r1 ; restore registers
reti
```

Digital Design — Chapter 8 — I/O Interfacing 76
Example: Gumnut Executive

```
;;; main program

main:       jsb init_interrupts
            enai

main_loop:  stby
            ldm r1, rtc_int_flag
            sub r0, r1, 1
            bnz main_next
            jsb task_2ms
            stm r0, rtc_int_flag

main_next:  ...
            jmp main_loop
```

- **Note**: `task_2ms` not called as part of interrupt handler
  - Would slow down response to other interrupts
Summary

- Transducers: sensors and actuators
  - Analog-to-digital and digital-to-analog converters
- Input and output devices
- Controllers
  - Input, output, control, and status registers
  - Autonomous controllers
- Buses: multiplexed, tristate, open-drain
  - Bus protocols: signals, timing, operations
Summary

- Serial transmission
  - NRZ, embedded clock
- Real-time software
  - Reacting to I/O and timer events
  - Polling, interrupts
- Real-time executives