

VHDL Samples:

The following documentation presents a few examples of VHDL code developed for various "private" projects by Thomas W. Gustin.

-- ipctrl.vhd

-- VHDL model that handles all of the IP transfer operations,
-- including the instantiation of a few control registers, as required.

-- The architecture and functionality of the IP8320 supports several of
-- the "standard" IP protocols. Specifically, the access chart for the
-- IP8320 is:

-- All 64 ID locations are readable, with only the first twelve locations
-- providing "real" data, as presented by the iddata.gdf file. The last
-- four locations in the first sixteen locations are always zeroes. This
-- same data pattern is repeated four times throughout the ID address space.

-- All 64 I/O locations are readable, with the first 48 being the 16-bit
-- ADC conversion results, one each for each ADC. The last (64th) location
-- is a write/read location for the control bits for a few options available
-- concerning the operations of the IP8320. Note that this latter location
-- does not need to be written to in order to generate ADC acquisition and
-- the automatic posting of conversion results. In the power-up default
-- mode, simultaneous acquisitions will run continuously, where the user only
-- needs to read the results (from I/O space or Memory space) as the application
-- requires. The remaining 15 unused I/O locations read as zeroes. The
-- power-up acquisition rate is that of the higher speed.

-- The first 48 memory locations are readable, providing exactly the same data
-- as that presented in the first 48 I/O locations (see above). All other
-- memory reads result in zeroes, except for the highest 64th location which
-- reads the four status bits (readback of the control bits), just like the
-- I/O read operations described above.

-- No data is provided during interrupt vector reads, should such a transfer
-- be attempted. The IP8320 does not generate any interrupts, and no interrupt
-- vector read should be executed by the carrier.

-- The IP8320 does not insert any wait cycles for any of the transfers. Even
-- though the ADC update rate occurs at a different rate, the data path contains
-- a re-synchronization register bank to the 8MHz rate. This permits the IP8320
-- to update all 48 locations, even during a read transfer, without inserting
-- a wait cycle. This means that the data could be different between the select
-- and acknowledgment cycles. However, the state of the data at the end of the
-- acknowledgment cycle is that which the carrier actually "receives". Note that
-- are two different sampling rates available for the IP8320 board (more later).

-- The IP8320 does support all carrier-inserted "hold" cycles, without limit.
-- Due to the high performance nature of the IP8320, it is recommended that the

-- user refrain from using low-performance carriers, if possible.

-- One of the "optional" control bits that can be modified by the user, if desired,
-- is that of the "ackall" bit. When this bit is set, the IP8320 will always
-- generate an acknowledgment to the carrier, even for illegal addresses and
-- unsupported or unused transfer types. This is a valuable feature for those
-- carriers and bus systems that don't have sufficient resources to terminate
-- a "hung" transfer due to the lack of an acknowledgment from an IP module.
-- The power-up default state for this bit is zero, where the IP8320 only provides
-- an acknowledgment when the address and transfer are used and legal types.

-- There are two control bits for the one, bidirectional, open drain status/control
-- line to the IP8320. This signal is a normally negated high state known as
-- n_ipstrobe, which connects to the "strobe*" line of the logic connector of
-- the carrier. Both of the control bits power-up to their default negated states
-- of zero. The first control bit enables the n_ipstrobe "OUTPUT" status function
-- that generates an asserted low signal during the registration of new conversion
-- results. This "signal" can be used to start the retrieval process by the
-- carrier of the new conversion results. The second control bit enables the
-- n_ipstrobe "INPUT" control function that provides a mechanism to "stop" the
-- conversion process. When the input control function is enabled, the negated
-- high state of the n_ipstrobe signal basically enables conversions to take place.
-- When the input signal is driven low, conversions are stopped and the state
-- machine is reset to zero. This feature provides two significant functional
-- benefits. It can be used to cause several IP8320 boards to convert all of
-- their channels simultaneously, not just by 48 channel boundaries provided by
-- a single IP8320. It also permits the carrier to retrieve all 48 locations of
-- data before permitting the IP8320 to gather another set of data by the
-- subsequent negation of the input n_ipstrobe. When both functions are used,
-- the external controlling logic would hold the n_ipstrobe signal low until
-- a new conversion was desired. After it releases the n_ipstrobe signal to
-- be pulled high, then the next falling edge of n_ipstrobe, would indicated that
-- the conversions have just been posted. The external controller has slightly less
-- than one conversion time in which to re-assert "low" the n_ipstrobe input
-- control signal in order to retrieve the last conversion results. This response
-- time is approximately 8.8uS when converting at the high rate of sampling
-- operations. Since a "perfect" carrier can retrieve all of the 48
-- locations in 12uS, and since new conversion results are available every 9.2uS,
-- the input control function is necessary for the retrieval of all data from a
-- single sampling if all 48 channels are used. For fast carriers, and for those
-- applications that are using only a portion of the channels available on the
-- IP8320, or for applications where simultaneously sampled data is not critical,
-- then the input control function of n_ipstrobe can be ignored. If the application
-- only cares about the data as it is reading it, then both the output status and
-- the input control features of n_ipstrobe can be ignored (leaving both of the
-- control bits in their default "zero" negated states).

-- Since the IP8320 can post data from 48 channels much faster than any standard
-- 8 MHz IP Carrier can possibly retrieve the data without over-writes occurring,
-- another control bit has been instantiated into the IP8320's control registers
-- to facilitate the slowing down of the IP8320 board by a factor of four. When
-- data bit #3 is set to a one, slowclkena is asserted which changes the DCLK
-- rate from 2.5MHz to 625KHz, which changes the per channel sampling rate from
-- 108,695 samples per second per channel to about 27,173 samples per second per
-- channel. This latter rate provides about 766nS per channel per read rate for
-- data retrieval purposes, a rate which the better IP Module carriers can hit

```

--      and sustain if one wishes to capture all of the data generated by the IP8320,
--      without stopping operations between data retrieval operations.

-- *****

ENTITY ipctrl IS
PORT (
  clk8m: IN BIT;           -- synchronous 8MHz system clock from IP carrier
  n_reset : IN BIT;       -- asserted low IP Module reset signal
  n_write : IN BIT;       -- asserted low write control from the IP carrier
  n_ioSEL : IN BIT;       -- asserted low I/O select control from the IP carrier
  n_memSEL: IN BIT;       -- asserted low Memory select from the IP carrier
  n_idSEL : IN BIT;       -- asserted low ID select control from the IP carrier
  n_intSEL: IN BIT;       -- asserted low Interrupt Vector select from the carrier
  a1:      IN BIT;        -- address bit #1 from the carrier
  a2:      IN BIT;        -- address bit #2
  a3:      IN BIT;        -- address bit #3
  a4:      IN BIT;        -- address bit #4
  a5:      IN BIT;        -- address bit #5
  a6:      IN BIT;        -- address bit #6
  d0:      IN BIT;        -- Data bit #0
  d1:      IN BIT;        -- Data bit #1
  d2:      IN BIT;        -- Data bit #2
  d3:      IN BIT;        -- Data bit #3
  regena:  IN BIT;        -- register enable control signal from the ctrl8320
                                -- VHDL model. Driven high during the one "DCLK"
                                -- clock cycle that registers the new ADC conversion
                                -- results for reading purposes. The n_ipstrobe signal
                                -- is driven low while this signal is high if it is
                                -- enabled via the assertion of the control bit for
                                -- statoutena (see below).
  slowclkena : INOUT BIT; -- I/O's last location a[6..1]=3F, d[3]
                                -- when high enables the slow rate of conversions by
                                -- changing DCLK from 2.5MHz to 625KHz
  ackallena : INOUT BIT;   -- I/O's last location a[6..1]=3F, d[0]
                                -- when high enables acknowledgment of all transfer
                                -- attempts, regardless of address legality
                                -- defaults to negated zero, ack only legal transfers
  statoutena : INOUT BIT;  -- I/O's last location a[6..1]=3F, d[1]
                                -- when high enables the open drain assertion low of
                                -- strbout when regena is asserted high to signal the
                                -- n_ipstrobe line that a conversion is complete.
  ctrlinena : INOUT BIT;  -- I/O's last location a[6..1]=3F, d[2]
                                -- when high enables the n_ipstrobe input signal's
                                -- asserted low state to remove the strtrena signal,
                                -- thereby stopping conversions.
  memzero: IN BIT;        -- asserted high signal from a comparator that indicates
                                -- that the extended address for memory transfers, as
                                -- presented on d[15..0], is NOT zero.
  hihalf : OUT BIT;       -- asserted high when the a[6..1] lines indicate that the
                                -- address is higher than 23d (18h or higher)
  idsel:   OUT BIT;       -- asynchronously asserted high during any portion of
                                -- an ID transfer, from select through acknowledgment,
                                -- It is used to change the data output multiplexer from
                                -- ADC data to ID data.
  regclr:  OUT BIT;       -- This asserted high signal asynchronously clears the

```

```

-- ADC's resynchronization registers to provide data of
-- "zeroes" for unused address locations. This occurs
-- for all locations above the lowest 48 for memory reads,
-- and for all 15 of the top 16 for I/O reads of the ADC
-- information.
strbout : INOUT BIT; -- Output signal that enables and disables an external
-- tristate gate driving the n_ipstrobe pin.
n_ipstrobe:IN BIT; -- State of the pin, used as described above for input
-- control
strtena: OUT BIT; -- Start enable used to cause the ADC sequence engine to
-- freerun or be controled by the n_ipstrobe input.
-- This signal is negated if n_ipstrobe is asserted low
-- by an external device (strbout = 0) and ctrlinena
-- is asserted high; otherwise it is asserted high.
ack : INOUT BIT; -- Synchronously driven high to couple output data to
-- the carrier for reads.
n_ack : INOUT BIT); -- Synchronously driven low to acknowledge the carrier
-- at the end of a transfer, to or from the IP8320.

END ipctrl; -- end of the port declaration area

ARCHITECTURE behavior OF ipctrl IS

-- Internal signal declaration area (automatic INOUT functionality within ipctrl.vhd)
SIGNAL memtrans :BIT; -- post memory select cycle
SIGNAL iotrans :BIT; -- post I/O select cycle
SIGNAL idtrans : BIT; -- post ID select cycle
SIGNAL regmemz :BIT; -- registered memzero signal for use during
-- post-select cycle on memory transfers.
SIGNAL wrtblk : BIT; -- synchronous signal that is asserted as a
-- result of the write event during an I/O
-- transfer while a hold cycle is being generated
-- by the carrier in order to ensure that only
-- one registration occurs, during the first hold
-- cycle only.

-- End of internal signal declaration area

BEGIN

-- *****

-- This simple statement develops the "HIHALF" signal that switches between the two
-- groups of 24 channels in the data path controller. It is asserted whenever
-- a[6..1]=18h (24d) or greater.

    hihalf <= '1' WHEN
        ((a6 = '0' AND a5 = '1' AND a4 = '1') OR a6 = '1')
    ELSE '0';

-- This next asynchronous statement resets the data resynchronization registers
-- to provide data equal to all zeroes for unused memory and I/O locations. The
-- memory locations include all extended memory locations as signaled by the
-- registered version of the memzero input signal. MEMZERO will only be low when
-- d[15..0]=0000h, which only occurs during the select cycle of the memory transfer.
-- Therefore, regclr = 1 during a memory transfer if a6=a5=1 or regmemz=1, and it
-- will also be asserted during any I/O transfer where a6=a5=1 except when

```

-- a6=a5=a4=a3=a2=a1=1 (the last I/O location). Note that all of the timing related
 -- qualifiers to this statement are already synchronized to the 8MHz clock, and that
 -- this output signal, although asynchronously derived, drives the synchronous clear
 -- input to the resynchronization registers such that the outputs will go to zero
 -- on the rising edge of the 8MHz clock if regclr is asserted high. Therefore, the
 -- existence of any noise or spikes on the output of regclr are irrelevant except at
 -- the time just prior to the rising edge of the clock, where signals are stable.

```
regclr <= '1' WHEN
  (n_reset = '0' -- when IP Module is reset or
  OR
  (n_memsel = '0' AND n_iosel = '1' AND n_intsel = '1' AND
  n_idsel = '1' AND memzero = '1')
  -- memory select cycle and extended memory address is not zero
  OR
  (n_memsel = '0' AND n_iosel = '1' AND n_intsel = '1' AND
  n_idsel = '1' AND memzero = '0' AND a6 = '1' AND a5 = '1')
  -- memory select cycle and upper sixteen locations of lower 64
  OR
  (memtrans = '1' AND regmemz = '1')
  -- memory post-select and extended memory address was not zero
  OR
  (memtrans = '1' AND regmemz = '1' AND a6 = '1' AND a5 = '1')
  -- memory post-select and upper sixteen location of lower 64 locations
  OR
  (n_memsel = '1' AND n_iosel = '0' AND n_intsel = '1' AND
  n_idsel = '1' AND a6 = '1' AND a5 = '1' AND
  (NOT(a6 = '1' AND a5 = '1' AND a4 = '1' AND a3 = '1' AND
  a2 = '1' AND a1 = '1'))))
  -- I/O select cycle and upper sixteen locations except last location
  OR
  (iotrans = '1' AND a6 = '1' AND a5 = '1' AND
  (NOT(a6 = '1' AND a5 = '1' AND a4 = '1' AND a3 = '1' AND
  a2 = '1' AND a1 = '1'))))
  -- I/O post-select and upper sixteen locations except last location
ELSE '0'; -- otherwise zero.
```

-- This next concurrent statement generates the "IDSEL" signal, which is asserted
 -- high during the valid ID select cycle or during the post-select transfer
 -- cycle(s) to permit the data path controller to switch from registered ADC data
 -- to the ID information to be placed onto the data bus. While no address
 -- qualifiers are used for this signal, the write control signal is monitored to
 -- ensure that this is a read transfer request.

```
idsel <= '1' WHEN
  ((n_memsel = '1' AND n_iosel = '1' AND n_intsel = '1' AND
  n_idsel = '0' AND n_write = '1')
  OR -- ID select cycle for read only
  (idtrans = '1' AND n_write = '1'))
ELSE '0'; -- otherwise zero.
```

-- *****

-- This simple process generates the flag bit that indicates that a valid memory
 -- transfer is taking place. It is asserted synchronously at the end of the
 -- select cycle, and is negated synchronously at the end of the acknowledgment

-- cycle after any interim hold cycles as determined by the negation of all
-- four select signals. Since only memory read operations can be performed
-- this process is qualified to the negated write control line.

```
membsy: PROCESS (clk8m, n_reset)
BEGIN
    IF n_reset = '0' THEN
        memtrans <= '0'; -- negate on IP Module reset condition
    ELSIF (clk8m'EVENT AND clk8m = '1') THEN -- on the rising edge of 8MHz clock:
        IF (n_write = '1' AND n_memsel = '0' AND n_iosel = '1' AND
            n_intsel = '1' AND n_idsel = '1' AND memzero = '0')
            THEN -- if a read transfer from only lower valid memory then
                memtrans <= '1'; -- synchronously assert at end of valid select
            ELSIF (n_ack = '0' AND n_memsel = '1' AND n_iosel = '1' AND
                n_intsel = '1' AND n_idsel = '1') THEN
                memtrans <= '0'; -- synchronously negate at end of transfer
            END IF;
        END IF;
    END PROCESS membsy;
```

-- This simple process generates the flag bit that indicates that a valid I/O
-- transfer is taking place. It is asserted synchronously at the end of the
-- select cycle, and is negated synchronously at the end of the acknowledgment
-- cycle after any interim hold cycles as determined by the negation of all
-- four select signals.

```
iobsy: PROCESS (clk8m, n_reset)
BEGIN
    IF n_reset = '0' THEN
        iotrans <= '0'; -- negate on IP Module reset condition
    ELSIF (clk8m'EVENT AND clk8m = '1') THEN -- on the rising edge of 8MHz clock:
        IF (n_memsel = '1' AND n_iosel = '0' AND n_intsel = '1' AND
            n_idsel = '1') THEN
            iotrans <= '1'; -- synchronously assert at end of valid select
        ELSIF (n_ack = '0' AND n_memsel = '1' AND n_iosel = '1' AND
            n_intsel = '1' AND n_idsel = '1') THEN
            iotrans <= '0'; -- synchronously negate at end of transfer
        END IF;
    END IF;
    END PROCESS iobsy;
```

-- This simple process generates the flag bit that indicates that a valid ID
-- transfer is taking place. It is asserted synchronously at the end of the
-- select cycle, and is negated synchronously at the end of the acknowledgment
-- cycle after any interim hold cycles as determined by the negation of all
-- four select signals. Since only ID read operations can be performed
-- this process is qualified to the negated write control line.

```
idbsy: PROCESS (clk8m, n_reset)
BEGIN
    IF n_reset = '0' THEN
        idtrans <= '0'; -- negate on IP Module reset condition
    ELSIF (clk8m'EVENT AND clk8m = '1') THEN -- on the rising edge of 8MHz clock:
        IF (n_write = '1' AND n_memsel = '1' AND n_iosel = '1' AND
            n_intsel = '1' AND n_idsel = '0') THEN -- if a read transfer then
            idtrans <= '1'; -- synchronously assert at end of valid select
        END IF;
    END PROCESS idbsy;
```

```

        ELSIF (n_ack = '0' AND n_memsel = '1' AND n_iosel = '1' AND
              n_intsel = '1' AND n_idsel = '1') THEN
            idtrans <= '0'; -- synchronously negate at end of transfer
        END IF;
    END IF;
END PROCESS idbsy;

-- This simple process develops the internal signal wrtblk, that flags the fact
-- that a write event has occurred during any I/O writes that may have inserted
-- hold cycles, in order to make sure that only a single write registration
-- happens during just the first hold cycle (further writes are block through
-- the termination cycle) for IP Module specification compliance.

hdblck: PROCESS (clk8m, n_reset)
BEGIN
    IF n_reset = '0' THEN -- negate during system reset
        wrtblk <= '0';
    ELSIF (clk8m'EVENT AND clk8m = '1') THEN -- rising edge of clock do:
        IF (iotrans = '1' AND n_write = '0' AND wrtblk = '0' AND
            n_iosel = '0') THEN -- first hold cycle event detection then
            wrtblk <= '1';
        ELSIF (wrtblk = '1' AND n_ack = '0' AND n_iosel = '1') THEN
            -- termination cycle detected then
            wrtblk <= '0';
        END IF;
    END IF;
END PROCESS hdblck;

-- *****

-- This simple process registers the memzero signal at the end of a memory select
-- cycle to "remember" the state of the extended memory address. An external
-- comparator looks at the 16 data lines and provides an asserted low signal on
-- memzero when all 16 lines are zeroes; otherwise it is high.

regzero: PROCESS (clk8m, n_reset)
BEGIN
    IF n_reset = '0' THEN
        regmemz <= '0'; -- negate on IP Module reset condition
    ELSIF (clk8m'EVENT AND clk8m = '1') THEN -- on the rising edge of 8MHz clock:
        IF (memzero = '1' AND memtrans = '0' AND n_memsel = '0' AND n_iosel = '1'
            AND n_intsel = '1' AND n_idsel = '1') THEN
            -- synchronously assert at end of valid memory select cycle
            -- if the extended memory on d[15..0] is NOT all zeroes.
            regmemz <= '1';
        ELSIF (n_ack = '0' AND n_memsel = '1' AND n_iosel = '1' AND
            n_intsel = '1' AND n_idsel = '1') THEN
            regmemz <= '0'; -- synchronously negate at end of transfer
        END IF;
    END IF;
END PROCESS regzero;

-- *****

-- This next process develops the control bit "ackallena", which is located at the
-- least significant bit of the most significant address of I/O space. The

```

-- "acknowledge all enable" signal, when asserted, enables the acknowledgment
 -- process to always generate an acknowledgment, regardless of transfer type.
 -- This control bit powers-up negated to zero. When left in this state, the
 -- n_ack signal generation will not occur for transfer attempts not supported by
 -- the IP8320.

```
aareg: PROCESS (clk8m, n_reset)
BEGIN
  IF n_reset = '0' THEN
    ackallena <= '0'; -- negate on IP Module reset condition
  ELSIF (clk8m'EVENT AND clk8m = '1') THEN -- on the rising edge of 8MHz clock:
    IF (iotrans = '1' AND n_write = '0' AND n_ack = '0' AND a6 = '1' AND
        a5 = '1' AND a4 = '1' AND a3 = '1' AND a2 = '1' AND a1 = '1' AND
        wrtblk = '0') THEN
      ackallena <= d0; -- write data bit #0 when address is correct
                        -- during an I/O write transfer
    END IF;
  END IF;
END PROCESS aareg;
```

-- This next process develops the control bit "statoutena", which is located
 -- at the bit #1 of the most significant address of I/O space. The
 -- "status output enable" signal, when asserted, enables the generation of a
 -- low state on the bidirectional n_ipstrobe signal line when the ADC registers
 -- are updated at the end of a conversion sequence. This is accomplished by
 -- the qualified control of the strbout signal.

```
soreg: PROCESS (clk8m, n_reset)
BEGIN
  IF n_reset = '0' THEN
    statoutena <= '0'; -- negate on IP Module reset condition
  ELSIF (clk8m'EVENT AND clk8m = '1') THEN -- on the rising edge of 8MHz clock:
    IF (iotrans = '1' AND n_write = '0' AND n_ack = '0' AND a6 = '1' AND
        a5 = '1' AND a4 = '1' AND a3 = '1' AND a2 = '1' AND a1 = '1' AND
        wrtblk = '0') THEN
      statoutena <= d1; -- write data bit #1 when address is correct
                        -- during an I/O write transfer
    END IF;
  END IF;
END PROCESS soreg;
```

-- This next process develops the control bit "ctrlinena", which is located
 -- at the bit #2 of the most significant address of I/O space. The
 -- "control input enable" signal, when asserted, enables the n_ipstrobe signal
 -- line state to control the start of conversions by its subsequent control
 -- of the strtena signal.

```
cireg: PROCESS (clk8m, n_reset)
BEGIN
  IF n_reset = '0' THEN
    ctrlinena <= '0'; -- negate on IP Module reset condition
  ELSIF (clk8m'EVENT AND clk8m = '1') THEN -- on the rising edge of 8MHz clock:
    IF (iotrans = '1' AND n_write = '0' AND n_ack = '0' AND a6 = '1' AND
        a5 = '1' AND a4 = '1' AND a3 = '1' AND a2 = '1' AND a1 = '1' AND
        wrtblk = '0') THEN
      ctrlinena <= d2; -- write data bit #2 when address is correct
    END IF;
  END IF;
END PROCESS cireg;
```

```

-- during an I/O write transfer
        END IF;
    END IF;
END PROCESS cireg;

-- This next process develops the control bit "slowckena", which is located
-- at the bit #3 of the most significant address of I/O space. The
-- "slow clock enable" signal, when asserted, enables the divide-by-four
-- prescaler in the ADC clock generation circuitry so that the ADC clock rate
-- changes from 2.5MHz to 625KHz.

slore: PROCESS (clk8m, n_reset)
BEGIN
    IF n_reset = '0' THEN
        slowckena <= '0'; -- negate on IP Module reset condition
    ELSIF (clk8m'EVENT AND clk8m = '1') THEN -- on the rising edge of 8MHz clock:
        IF (iotrans = '1' AND n_write = '0' AND n_ack = '0' AND a6 = '1' AND
            a5 = '1' AND a4 = '1' AND a3 = '1' AND a2 = '1' AND a1 = '1' AND
            wrtblk = '0') THEN
            slowckena <= d3; -- write data bit #3 when address is correct
            -- during an I/O write transfer
        END IF;
    END IF;
END PROCESS slore;

-- *****

-- This simple process develops the start enable signal for the ADC controller
-- module. It defaults to the asserted high state on power-up to make sure the
-- ADCs power-up converting all channels automatically. The user has the option
-- of slaving this IP8320 to an external controller via the assertion of the
-- control bit ctrlina. If this latter bit is set, then the strtena bit follows
-- the n_ipstrobe input, as synchronized by the clk8m clock signal. Further, it is
-- qualified against this IP8320's strobe output driver signal to make sure that the
-- signal being detected as asserted low on the n_ipstrobe pin is not being driven
-- low by this IP8320's output pulse indicating that a conversion is complete.

enarun: PROCESS (clk8m, n_reset)
BEGIN
    IF n_reset = '0' THEN
        strtena <= '1'; -- assert on power up and other resets to run automatically
    ELSIF (clk8m'EVENT AND clk8m = '1') THEN -- otherwise, on the clock's up edge:
        IF (ctrlina = '0' OR (ctrlina = '1' AND n_ipstrobe = '1')) THEN
            -- run conversions if control is not needed or if it
            strtena <= '1'; -- is needed, when the strobe input is undriven high
        ELSIF (ctrlina = '1' AND n_ipstrobe = '0' AND strbout = '0') THEN
            -- stop conversions if control is needed and the strobe
            strtena <= '0'; -- signal pin is low and this IP8320 is not driving it
        END IF;
    END IF;
END PROCESS enarun;

-- This simple port driver drives high the tristate control pin for the open
-- drain output driver for the n_ipstrobe signal pin when regena is asserted high
-- while new ADC data is being registered provided it is enabled by the assertion

```

-- of the control bit statoutena.

```
strbout <= '1' WHEN -- drive n_ipstrobe low,
    (regena = '1' AND statoutena = '1')
ELSE '0'; -- otherwise tristate the n_ipstrobe output driver.
```

-- *****

-- This next process develops the asserted low acknowledgment signal for control
-- handshaking with the IP8320's carrier. Since there are no wait cycles inserted
-- by the IP8320, n_ack is always asserted low following a VALID select cycle and
-- then negated after any carrier-inserted hold cycles, or right away if no hold
-- cycles are requested. If the control bit ackallena is set by the user, then
-- n_ack is always asserted following ANY select cycle, and negated as described
-- above. When ackallena is negated (default) then ID write attempts, memory write
-- attempts, read of memory attempts above the first 64 locations, and any interrupt
-- vector transfer attempts will all be met with a non-asserted acknowledgment which
-- can cause some computer bus types to "hang" due to the lack of a response from
-- this IP8320 module. If this is possible on the user's platform, then it is
-- recommended that the user set the control bit ackallena, at least until all
-- application software is debugged. Normal usage of this IP8320 module should
-- permit the qualified acknowledgment to be sufficient for operations.

```
nackgen: PROCESS (clk8m, n_reset)
BEGIN
```

```
    IF n_reset = '0' THEN n_ack <= '1'; -- negate high on reset, else
    ELSIF (clk8m'EVENT AND clk8m = '1') THEN -- on the rising edge of clock then
        IF ((n_write = '1' AND n_memsel = '0' AND n_iosel = '1' AND
            n_intsel = '1' AND n_idsel = '1' AND memzero = '0' AND ackallena = '0')
            -- if a memory read transfer to valid space only, or
            OR
            (n_memsel = '1' AND n_iosel = '0' AND n_intsel = '1' AND
            n_idsel = '1' AND ackallena = '0')
            -- if an I/O read or write transfer, or
            OR
            (n_memsel = '1' AND n_iosel = '1' AND n_intsel = '1' AND
            n_idsel = '0' AND n_write = '1' AND ackallena = '0')
            -- if an ID read only transfer, or
            OR
            (ackallena = '1' AND (n_memsel = '0' OR n_intsel = '0' OR n_idsel = '0'
            OR n_iosel = '0'))) THEN -- any transfer if acknowledge all is enabled
            n_ack <= '0'; -- THEN assert low "n_ack"
        ELSIF (n_ack = '0' AND n_memsel = '1' AND n_iosel = '1' AND n_intsel = '1'
            AND n_idsel = '1') THEN -- after hold cycles then
            n_ack <= '1';
        END IF;
    END IF;
```

```
END PROCESS nackgen; -- end of the n_ack generator
```

-- *****

-- This last process develops the "ack" signal which drives the output drivers of the
-- data bus to acknowledge a read request with data to the carrier. It is asserted
-- high during the same time n_ack is asserted low for valid read only transfers, and
-- it is negated as the n_ack signal is negated.

```

ackgen: PROCESS (clk8m, n_reset)
BEGIN
    IF n_reset = '0' THEN ack <= '0'; -- negate low on reset, else
    ELSIF (clk8m'EVENT AND clk8m = '1') THEN -- on the rising edge of clock then
        IF ((n_write = '1' AND n_memsel = '0' AND n_iosel = '1' AND
            n_intsel = '1' AND n_idsel = '1' AND memzero = '0')
            -- if a memory read transfer to valid space only, or
            OR
            (n_memsel = '1' AND n_iosel = '0' AND n_intsel = '1' AND
            n_idsel = '1' AND n_write = '1')
            -- if an I/O read transfer, or
            OR
            (n_memsel = '1' AND n_iosel = '1' AND n_intsel = '1' AND
            n_idsel = '0' AND n_write = '1')) THEN
            -- if an ID read only transfer, or
            ack <= '1'; -- THEN assert low "n_ack"
        ELSIF (ack = '1' AND n_memsel = '1' AND n_iosel = '1' AND n_intsel = '1'
            AND n_idsel = '1') THEN -- after hold cycles then
            ack <= '0';
        END IF;
    END IF;
END PROCESS ackgen; -- end of the ack signal generator

-- *****
end behavior;

```

```

-- state.vhd

```

```

-- This VHDL model develops the next state information for the register
-- LPM on the CTRL8320.gdf schematic, and it develops several key control
-- signals for the capturing of ADC data on the IP8320.

```

```

-- The core of the main sequential engine's design is that of a 23 state,
-- "mock-Gray" code engine. Since it contains 23 states, there is one
-- state transition that experiences a bit change for two bits instead of
-- just one. Furthermore, this model is fully asynchronous in nature.
-- The external 5 bit Register instantiated as an LPM_DFF provides the
-- memory for the sequential state machine operations. Additionally,
-- since only one bit changes at a time with the one noted exception,
-- system control signals can be state-derived without further registration
-- based upon the state of the registered information from the engine.

```

```

-- An external 5MHz oscillator's signal is gated by reset and a start enable
-- signal and is divided by two to develop "DCLK", the 2.5MHz data clock for
-- the 48 ADCs and for the data capturing logic within this Altera part.
-- When "slowclkssel" is asserted high by a software control bit selection, then
-- "DCLK" is driven at a 625KHz data clock rate. This not only reduces the
-- sampling rate 1/4th (to about 27Ks/s/c) but it permits four times the amount
-- of time for data retrieval, which is the intention of this feature. The

```

```
-- normal amount of time to remove the last conversion results from memory
-- before they are over-written with the next sampling is 9.2uS, clearly
-- too short of a time for all standard carriers if all 48 channels are being
-- used. When "slowcksel" is enable, 36.8uS exists between conversions for
-- data retrieval. This provides an average of 766nS per read of the 48 ADC
-- conversion results, where some of the better carriers can sustain a read
-- rate of one per 750nS. This feature only needs to be invoked if all of the
-- data retrieved needs to be from a single sampling and all 48 channels are
-- being simul-sampled.
```

```
-- There are 22 clock states needed to perform an acquisition, including the
-- retrieval of data. The 23rd state recycles the ADCs before starting the
-- next simultaneous acquisition on all channels.
```

```
-- When conversions are not actively being executed, as indicated by the
-- "strtena" input signal, then all 48 ADCs are held in "shutdown" mode by
-- the negation high of the "n_cs" signal.
```

```
-- *****
```

```
ENTITY state IS
```

```
PORT (
```

```
    clk5m   : IN BIT;           -- synchronous ADC clock source (2 times)
    n_reset : IN BIT;           -- system reset asserted low
    strtena  : IN BIT;           -- start enable when high
    slowckena : IN BIT;         -- switch input that further divides the ADC
                                -- clock rate down by four (2.5MHz -> 625KHz)
    current  : IN BIT_VECTOR(4 downto 0); -- current state output
    dclk     : INOUT BIT;        -- 2.5MHz ADC synchronization & data clock
    nexttd   : OUT BIT_VECTOR(4 downto 0); -- next state data
    n_cs     : OUT BIT;          -- asserted low common ADC chip select
                                -- negated high = shutdown mode
    shftena  : OUT BIT;          -- shift register shift enable
    regena   : OUT BIT);         -- Register enable
```

```
END state;
```

```
ARCHITECTURE behavior OF state IS
```

```
-- internal signals area:
-- SIGNAL clk2m5 : BIT; -- 2.5MHz from 5MHz input
-- SIGNAL clk1m25 : BIT; -- 1.25MHz from the clk2m5 signal above
-- SIGNAL clk625k : BIT; -- 625KHz from the 1.25MHz signal above
-- end of internal signals
```

```
BEGIN
```

```
-- *****
```

```
-- The "mock" gray-code sequence with 23 states for this design is:
-- 0, 1, 3, 2, 6, 7, 5, 4, C, D, F, E, A, B, 9, 8, 18, 19, 1B, 13, 17,
-- 15, 11, and back to 0. Notice that all state changes involve single
-- bit changes except for the last one when the count goes from 11h -> 00h.
-- Also note that the "normal" gray-code sequence for a 5-bit counter is no
-- longer followed after state = 1Bh.
```

```

nextd <=      "00001" WHEN
              (current = "00000" AND strtena = '1')      -- 0 -> 1
              ELSE  "00011" WHEN current = "00001" -- 1 -> 3
              ELSE  "00010" WHEN current = "00011" -- 3 -> 2
              ELSE  "00110" WHEN current = "00010" -- 2 -> 6
              ELSE  "00111" WHEN current = "00110" -- 6 -> 7
              ELSE  "00101" WHEN current = "00111" -- 7 -> 5
              ELSE  "00100" WHEN current = "00101" -- 5 -> 4
              ELSE  "01100" WHEN current = "00100" -- 4 -> C
              ELSE  "01101" WHEN current = "01100" -- C -> D
              ELSE  "01111" WHEN current = "01101" -- D -> F
              ELSE  "01110" WHEN current = "01111" -- F -> E
              ELSE  "01010" WHEN current = "01110" -- E -> A
              ELSE  "01011" WHEN current = "01010" -- A -> B
              ELSE  "01001" WHEN current = "01011" -- B -> 9
              ELSE  "01000" WHEN current = "01001" -- 9 -> 8
              ELSE  "11000" WHEN current = "01000" -- 8 -> 18
              ELSE  "11001" WHEN current = "11000" -- 18 -> 19
              ELSE  "11011" WHEN current = "11001" -- 19 -> 1B
              ELSE  "10011" WHEN current = "11011" -- 1B -> 13
              ELSE  "10111" WHEN current = "10011" -- 13 -> 17
              ELSE  "10101" WHEN current = "10111" -- 17 -> 15
              ELSE  "10001" WHEN current = "10101" -- 15 -> 11
              ELSE  "00000";

```

-- Note that the double bit change occurs in the sequence of events in such
-- a manner that no control signals are involved with these states so that
-- no glitches can be accidentally generated.

-- The "shftena" (serial shift enable) occurs during states 2 through 1B,
-- and the registration is enabled ("regena" asserted) during state = 13.

-- Note that once a sequence is started, it will complete the sequence even
-- if the start enable signal is negated during the sequence. This important
-- control signal is only sampled at the beginning of the sequence.

-- *****

-- This section develops the synchronous clock "DCLK", including its selectable
-- slow clock prescaler signals.

-- This first little process develops the 2.5MHz clock that is used as DCLK
-- for high speed acquisition operations.

```

divby2: PROCESS (clk5m, n_reset)
BEGIN
    IF n_reset = '0' THEN -- during reset, turn off clock
        clk2m5 <= '0';
    ELSIF (clk5m'EVENT AND clk5m = '1') THEN -- on the rising edge of 5MHz clock
        clk2m5 <= NOT(clk2m5);
    END IF;
END PROCESS divby2; -- end of 2.5MHz clock generator

```

-- This next little process develops the 1.25MHz clock that is used to develop
-- the slow speed clock for low speed acquisition operations.

```

divby4: PROCESS (clk2m5, n_reset)
BEGIN
    IF n_reset = '0' THEN -- during reset, turn off clock
        clk1m25 <= '0';
    ELSIF (clk2m5'EVENT AND clk2m5 = '1') THEN -- on the rising edge of 2.55MHz clock
        clk1m25 <= NOT(clk1m25);
    END IF;
END PROCESS divby4; -- end of 1.25MHz clock generator

-- This little process develops the 625KHz clock that is used as DCLK
-- for low speed acquisition operations.

divby8: PROCESS (clk1m25, n_reset)
BEGIN
    IF n_reset = '0' THEN -- during reset, turn off clock
        clk625k <= '0';
    ELSIF (clk1m25'EVENT AND clk1m25 = '1') THEN -- on the rising edge of 1.25MHz clock
        clk625k <= NOT(clk625k);
    END IF;
END PROCESS divby8; -- end of 625KHz clock generator

-- This next simple statement develops the "dclk" for the ADCs and several
-- functional blocks inside this Altera device, from a 5MHz oscillator input,
-- as qualified by the negation of the global reset signal and the assertion
-- of the strtena (start enable signal). DCLK is either 2.5MHz or it is 625KHz
-- depending upon the state of the "slowclkena" switch input, as described in
-- the port declaration area.

    dclk <= clk2m5 WHEN
        (n_reset = '1' AND strtena = '1' AND slowclkena = '0')
    ELSE clk625K WHEN
        (n_reset = '1' AND strtena = '1' AND slowclkena = '1')
    ELSE '0';

-- *****

-- This next simple signal assignment develops the asserted high pulse to
-- register the just captured serial data into the parallel read locations.
-- It occurs at the end of the "shftena" asserted cycle.

    regena <= '1' WHEN current = "10011" -- assert during state =13
    ELSE '0';

-- This next simple signal assignment generates the asserted low "n_cs"
-- signal to the ADCs, which happens to coincide with the assertion of
-- regena, as above.

    n_cs <= '1' WHEN (current = "10011" OR strtena = '0')
        -- negate during state = 13 or when starts are disabled
        -- The single cycle negation prepares the ADCs for a new
        -- sequence on the next clock cycle when n_cs is again
        -- asserted low.
    ELSE '0';

-- *****

```

```
-- This next statement develops the asserted high "shftena" (shift register
-- enable) signal during engine states of:
-- 2, 6, 7, 5, 4, C, D, F, E, A, B, 9, 8, 18, 19, and 1B.
```

```
shftena <= '0' WHEN -- negate during states:
    (current = "10011"      -- state = 13
    OR
    current = "10111"      -- state = 17
    OR
    current = "10101"      -- state = 15
    OR
    current = "10001"      -- state = 11
    OR
    current = "00000"      -- state = 00
    OR
    current = "00001"      -- state = 01
    OR
    current = "00011")    -- state = 03
ELSE '1'; -- otherwise assert
```

```
END behavior;
```

```
-- VHDL version of "triple 32-times-32 Multiplier" that is also
-- constructed using LPM's in the Multex1.gdf design file
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
```

```
ENTITY multex2 IS
```

```
    PORT
    (
    ad, bd, cd :      IN unsigned (31 downto 0);
                    -- 3 32-bit data input buses
    axb, bxc, axc :  OUT unsigned (63 downto 0));
                    -- 3 64-bit result output buses
```

```
END multex2;
```

```
ARCHITECTURE behavior OF multex2 IS
```

```
BEGIN
```

```
    axb <= (ad * bd);      -- "A" data bus times "B" data bus
    bxc <= (bd * cd);      -- "B" data bus times "C" data bus
    axc <= (ad * cd);      -- "A" data bus times "C" data bus
```

```
END behavior;
```



```

-- ***** implementation notes *****

-- On reset=high, the Gray7 counter is asynchronously reset to
--           Gray=0, and is held there for the duration of the reset assertion
-- Regardless of the state of "Pause", on the first rising edge of the
--           clock input when "StrtEna" is also high, the counter will be
--           advanced to the Gray=1 state
-- The counter will continuously count as long as "Pause" remains negated
--           low and as long as "StrtEna" is asserted high during the Gray=0 state

-- Truth Table For Inputs:                Output Operation:
-- Reset StrtEna Pause Gray=             Gray=0 while reset=1
-- 1     x     x     x                   hold at Gray=0
-- 0     0     x     0                   next clock Gray->1
-- 0     1     x     0                   next clock Gray->3,2,6,7,5,0
-- 0     x     0     1,3,2,6,7,5       hold at Gray=1,3,2,5,7, or 5
-- 0     x     1     1,3,2,6,7,5

-- If the external circuitry that is using this sequential state machine needs
-- to have an uninterrupted flow of counts (a full sequence of 0,1,3,2,6,7,5,0)
-- then the single cycle assertion of "strtena" will work provided "pause"=0.
-- If the external circuitry needs to be able to control on a cycle-by-cycle
-- basis when the next gray code count is permitted to occur, then throttling
-- by the "pause"=1 control signal is used. The conditions for pause=0, which
-- allows for gray code advancement to occur, is developed externally based upon
-- the system signals that qualify the valid state for this engines movement.

architecture BEHAVIOR of Gray7 is
-- Internal signals:

    signal next_gray: STD_LOGIC_VECTOR (2 downto 0);

BEGIN

-- This process is a qualified synchronous registration statement that
-- converts the "next_gray" D-inputs to the "now_gray" Q-outputs.

reg3: PROCESS (clk, rst) -- execute on any change in "clk" or "rst"
BEGIN
    If rst = '1' THEN
        graycnt <= "000"; -- clear the count to Gray=0
    ELSIF (clk'EVENT AND clk = '1') THEN -- on the rising edge of clk
        IF ((graycnt = "000" AND strtена = '1' AND rst = '0') OR
            (graycnt /= "000" AND pause = '0' AND rst = '0')) THEN
            graycnt <= next_gray; -- if counting permitted, increment gray value
        END IF;
    END IF;
END PROCESS reg3; -- end of the 3-bit register

```

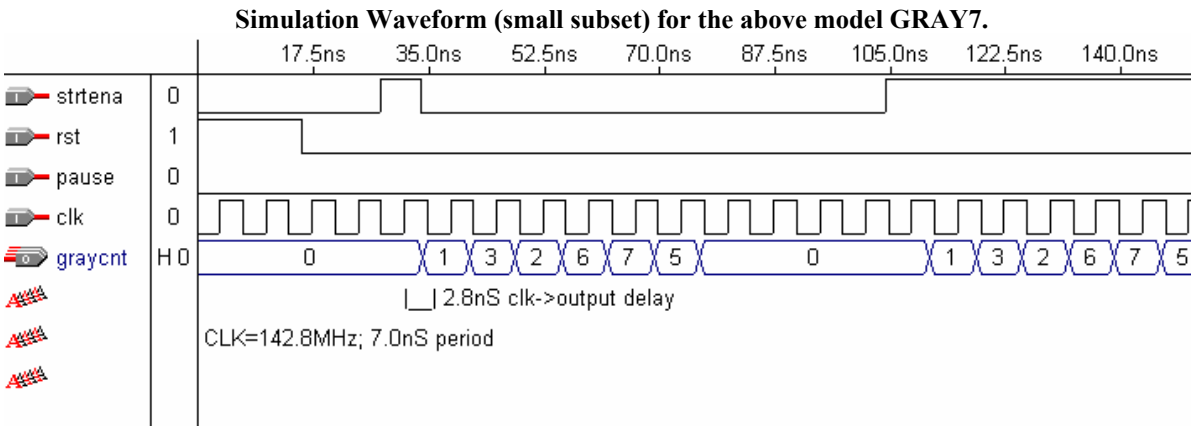
```
-- This next concurrent statement determines what the D-inputs are to the
-- above registers based solely on the current Q-outputs on "GRAYCNT"
```

```
    next_gray <= "001" WHEN graycnt = "000"           -- 0 -> 1
                ELSE "011" WHEN graycnt = "001"       -- 1 -> 3
                ELSE "010" WHEN graycnt = "011"       -- 3 -> 2
                ELSE "110" WHEN graycnt = "010"       -- 2 -> 6
                ELSE "111" WHEN graycnt = "110"       -- 6 -> 7
                ELSE "101" WHEN graycnt = "111"       -- 7 -> 5
                ELSE "000"; -- 5 -> 0 and so does illegal state 4

END BEHAVIOR;           -- end of Gray7
```

```
--Targetting Notes:
```

```
-- Chip/           Input Output Bidir   Shareable
-- POF   Device    Pins  Pins  Pins  LCs  Expanders  % Utilized
-- gray7 EPM7032SLC44-5 4    0    3    3    0    9 %
-- Clock to Output delay = 2.8ns worst case; 250MHz maximum (clock spec)
```



The above VHDL code represents the only examples that were not written for a customer or former employer.