



PERDYN11088™

ALTERA® AVALON® BUS WRAPPER

Application Note

This application note illustrates the use of the PERDYN11088™ Keypad Interface IP with an Altera® Nios® II soft processor system via the Avalon bus. An Avalon memory-mapped slave 'wrapper' allows the PERDYN11088 IP core to be added to the Nios II processor using the Qsys software tool. The Verilog® wrapper file and Nios® II C code are available to PERDYN11088® licensees to serve as a basis in their own Nios® II designs.

INTRODUCTION

The PERDYN11088 Keypad Interface IP module has the flexibility to operate in a wide variety of embedded systems, providing a convenient interface between operator touch controls and nearly any microprocessor. Traditional 'hard' microcontrollers connect with the PERDYN11088 IP implemented in a FPGA or CPLD, forming a 'processor companion' chip in which one or more microcontroller peripherals are created, giving enhanced functionality, speed, and performance to the host microcontroller.

This application note illustrates an alternate use of the keypad peripheral in which the processor is synthesized in FPGA logic fabric alongside the keypad peripheral. So called 'soft processors' are utilized for a variety of reasons including obsolescence-proofing hardware & firmware and providing a tight coupling between processor and peripheral. We will utilize the Nios II/e soft processor, available without license fee from Altera Corporation. According to Altera, the Nios II processor is the most utilized soft processor in the world, and design support for Nios II based systems is available from Altera (www.altera.com) as well as the online user forum (www.alteraforum.com).

The Nios II processor can utilize several bus structures, however it is often used with the Avalon bus. The Avalon interface simplifies system design by allowing simple connection of components inside the FPGA. Several modes are available, including a streaming interface (Avalon-ST) for peripherals requiring high data throughput, and a memory-mapped interface (Avalon-MM) for peripherals that require only occasional data transfer. The following diagram illustrates the architecture of a basic Nios II processor system.

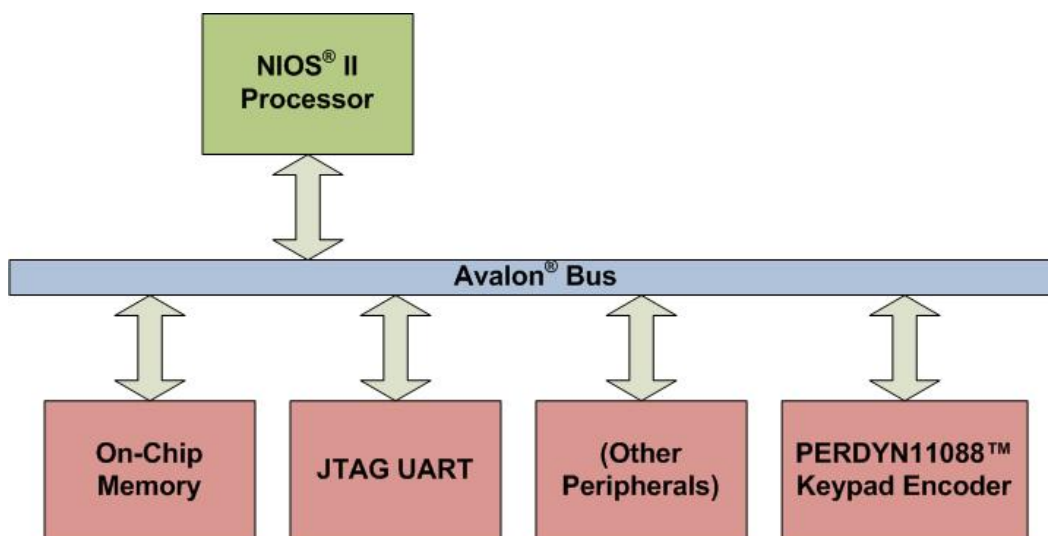


Figure One: Block Diagram of a Basic Nios® II Embedded Processor System

Altera's Quartus II software generates an application-specific Avalon bus suited to the data-flow requirements of the particular embedded system being designed into the FPGA. Through the use of Altera's Qsys utility, elements of the embedded system are connected to the Avalon bus using only a few mouse clicks, creating all the required data, clock, and reset connections graphically. Peripheral memory-mapped addresses, interrupt numbers, and other system details are easily handled using the Qsys tool, generating an embedded system that incorporates all the desired peripherals via the Avalon bus, making them easily accessible to the processor's firmware.

The following image illustrates the Qsys tool being used to create a Nios II embedded system incorporating the PERDYN11088 Keypad Interface IP. Notice the Avalon bus connections along the left side, and the peripheral base address and interrupt number to the right of the PERDYN11088 peripheral.

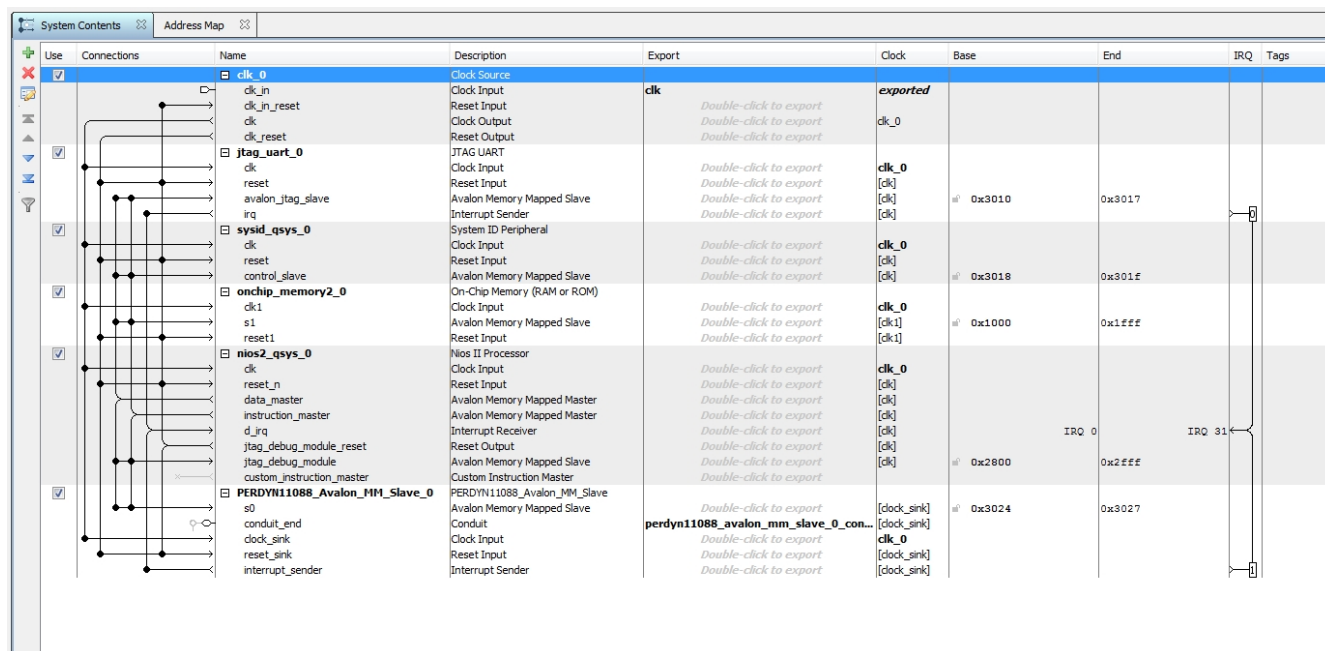


Figure Two: Qsys Processor System

To connect the PERDYN11088™ Keypad Interface IP to the Nios II processor via the Avalon bus, Peripheral Dynamics has created a 'wrapper' that converts the PERDYN11088™ data signals into an interface suitable for the Avalon bus. Our PERDYN11088™ IP has a six-bit data output ('keyData') as well as a data ready strobe ('dataValid'). This Avalon wrapper, written in Verilog, captures keypress data from the PERDYN11088 IP and generates the signals necessary for it to interact on the Avalon bus. Implemented as a memory-mapped slave peripheral (Avalon-MM), the following signals are required to provide data to the Nios II processor:

read – a 1-bit input signal that the processors uses to access keypress data from the wrapper.

readdata – a 32-bit data line that the 6 bits of keypress data are communicated through.

irq0_irq – this output, generated by the wrapper, provides an interrupt signal to the processor, informing it that a new keypress has occurred.

Name	Interface	Signal Type	Width	Direction
avs_s0_read	s0	read	1	input
avs_s0_readdata	s0	readdata	32	output
row	conduit_end	export	8	input
column	conduit_end	export	8	output
clk	clock_sink	clk	1	input
reset	reset_sink	reset	1	input
ins_irq0_irq	interrupt_sender	irq	1	output

Figure Three: Wrapper Interface Signals

As illustrated above, the 'ins_irq0_irq' interrupt signal provides an interrupt to the Nios II processor when a new keypress occurs. The wrapper generates this interrupt signal when the PERDYN11088 IP strobos dataValid, indicating that a new keypress has been detected. The interrupt signal is held high until the processor accesses the peripheral with a read command.

The Verilog code for the PERDYN11088 Avalon bus wrapper is listed below. Notice that this wrapper instantiates a clock divider that creates the 1 MHz clock required by the IP, and also instantiates the PERDYN11088 IP itself. It also generates the Avalon bus signals described above.

```
// PERDYN11088 Keypad Interface IC Avalon MM Slave Wrapper
module PERDYN11088_Avalon_MM_Wrapper (
    row,                                // declare as wrapper conduit
    column,                              // declare as wrapper conduit
    clk,                                 // Avalon MM Slave signal
    reset,                               // Avalon MM Slave signal
    avs_s0_read,                         // Avalon MM Slave signal
    avs_s0_readdata,                    // Avalon MM Slave signal
    ins_irq0_irq                         // Avalon MM Slave signal
);

// declare IO direction
input [7:0] row;
output [7:0] column;
input clk;
input reset;
input avs_s0_read;
```

```

output [31:0] avs_s0_readdata;
output ins_irq0_irq;

// declare module port types
wire avs_s0_read;
reg [31:0] avs_s0_readdata;
reg ins_irq0_irq;           // signal to level-sensitive Nios II ISR handler (internal interrupt controller)
wire clk;
wire reset;
wire [7:0] row;
wire [7:0] column;
wire [5:0] keyData;        // input vector from PERDYN11088 IP
wire dataValid;           // keyData latch from PERDYN11088 IP

// declare local variables
reg [1:0] risingData;      // when this is {0,1} activate irq signal until cleared by ISR read
wire divClk;

// Instantiate Clock Divider
programmableClockDivider U2 (
    .Clk_in (clk),
    .Clk_out (divClk)
);

// Instantiate IP Block
PERDYN11088 U1 (
    .oneMHz_clk (divClk),
    .reset_n (~reset),
    .row (row),
    .column (column),
    .keyData (keyData),
    .dataValid (dataValid)
);

always @ (posedge clk or posedge reset)
begin
    if (reset == 1'b1) // assume active-high reset
        begin
            ins_irq0_irq <= 1'b0;
            avs_s0_readdata <= 32'b0; // clear data bus driver
            risingData[1:0] <= 2'b0;
        end
    else
        begin
            risingData[1:0] <= {risingData[0],dataValid}; // shift in dataValid to find rising edge
            if ( risingData == 2'b01) // new keypress
                begin

```

```

        avs_s0_readdata <= {26'b0,keyData};
        ins_irq0_irq <= 1'b1;           // enable interrupt
    end
    else if (avs_s0_read == 1'b1)     // taken when IO location is read by Avalon bus
    begin
        ins_irq0_irq <= 1'b0;       // disable interrupt
    end
    end                                 // end non-reset path
end                                     // end always@
endmodule

```

Nios II PROCESSOR C CODE

The heart of the PERDYN11088's Avalon bus interface is the C program's interrupt service routine (ISR). This ISR is called when the bus wrapper activates the Nios II interrupt request (irq) signal, notifying the processor that it should branch to its interrupt routine and service the interrupt. The ISR C code is listed below:

```

static void irqhandler(void* context)
{
    readBack = IORD_32DIRECT (PERDYN11088_AVALON_MM_SLAVE_0_BASE,0);
    printFlag++;
    usleep(10000);           // hold here 10ms while interrupt signal clears
}

```

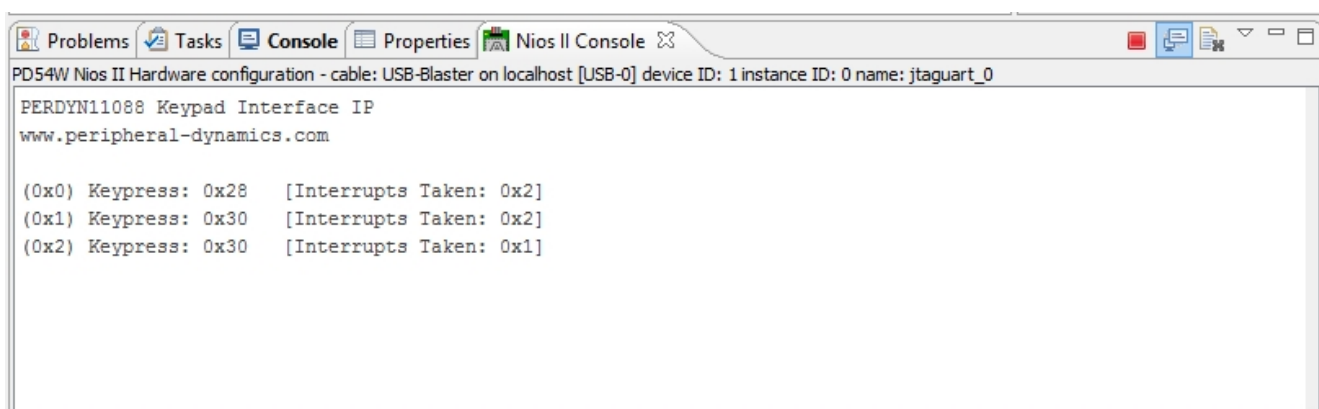
When the PERDYN11088 receives a keypress it outputs the keypress data on its keyData port, which in turn is captured by the Avalon wrapper. This wrapper creates a memory-mapped register that the Nios II processor can read. The wrapper memory location is read using the C function IORD_32DIRECT, which reads a 32 bit value from the memory address assigned to the wrapper in Qsys. It should also be noted that the Avalon wrapper Verilog code is designed so that a Nios II memory access to the wrapper address clears the interrupt which triggered the read.

The main loop of the example C program awaits a call to the interrupt service routine and displays it on the Quartus II JTAG console when received. The main loop of the C code is shown below:

```
while (1) {  
  if (printFlag != 0) {  
    alt_printf("(0x%x) Keypress: 0x%x [Interrupts Taken: 0x%x]\n",pressCount++,readBack,printFlag);  
    printFlag = 0;  
    usleep(50000);  
  }  
}
```

As can be seen in the code above, when the ISR is called it increments the variable 'printFlag'. When the main loop sees this flag, it proceeds to print the data and clear the flag. In a full embedded system, the processor would be tasked with other chores, and would process key-presses received via the ISR as they occur.

The main loop and ISR work together to provide the following console output when keys are pressed:



```
PD54W Nios II Hardware configuration - cable: USB-Blaster on localhost [USB-0] device ID: 1 instance ID: 0 name: jtaguart_0  
PERDYN11088 Keypad Interface IP  
www.peripheral-dynamics.com  
  
(0x0) Keypress: 0x28 [Interrupts Taken: 0x2]  
(0x1) Keypress: 0x30 [Interrupts Taken: 0x2]  
(0x2) Keypress: 0x30 [Interrupts Taken: 0x1]
```

Figure Four: JTAG UART Console Output

Notice that occasionally the flag variable is incremented twice prior to the main loop processing the keypress data. This is caused by a relatively brief ISR period combined with the varying mechanical bounce period of the key contacts. Essentially, the ISR completes prior to the interrupt signal being cleared, and consequently the ISR may be taken a second time. A slight delay has been added to reduce this effect, and if it poses a problem in your particular application the interrupts may be manually disabled & re-enabled for a period longer than the brief ISR. For additional details on Nios II spurious interrupts caused by fast ISR latency please see Chapter 8 of Altera's *Nios II Interrupt Service Routines* document (n2sw_nii52006.pdf).

DETAILS OF THE CUSTOM PERIPHERAL

Although the PERDYN11088 Avalon bus wrapper may be used without looking at the details of its implementation, some users may appreciate a 'behind the scenes' look at the creation and timing of the wrapper interface signals. This section presents a brief overview of the Qsys utility's custom component generation tool.

Selecting 'New Component' from the Qsys file menu launches the custom component generation tool. This utility imports and processes, in this case, a Verilog file, analyzing it for signals that belong on the Avalon bus. Proper naming convention helps the auto-recognition feature identify signals appropriately. The screen capture below shows the Component Editor's file analysis screen.

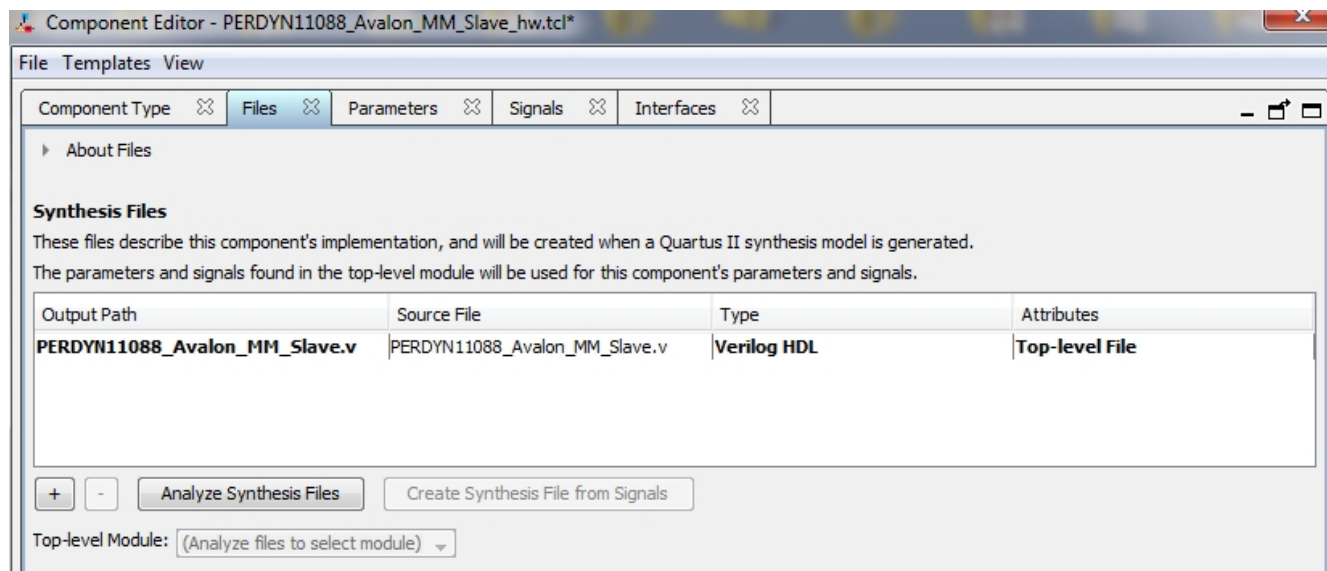


Figure Five: Custom Peripheral File Analysis Screen

Note that the wrapper's Verilog source file, which instantiates the PERDYN11088 IP, is selected as the top-level source file. The wrapper's code is analyzed, and the various signals forming the wrapper's interface to the Avalon bus are identified. The following screen capture shows the read interface timing that the wrapper is configured for.

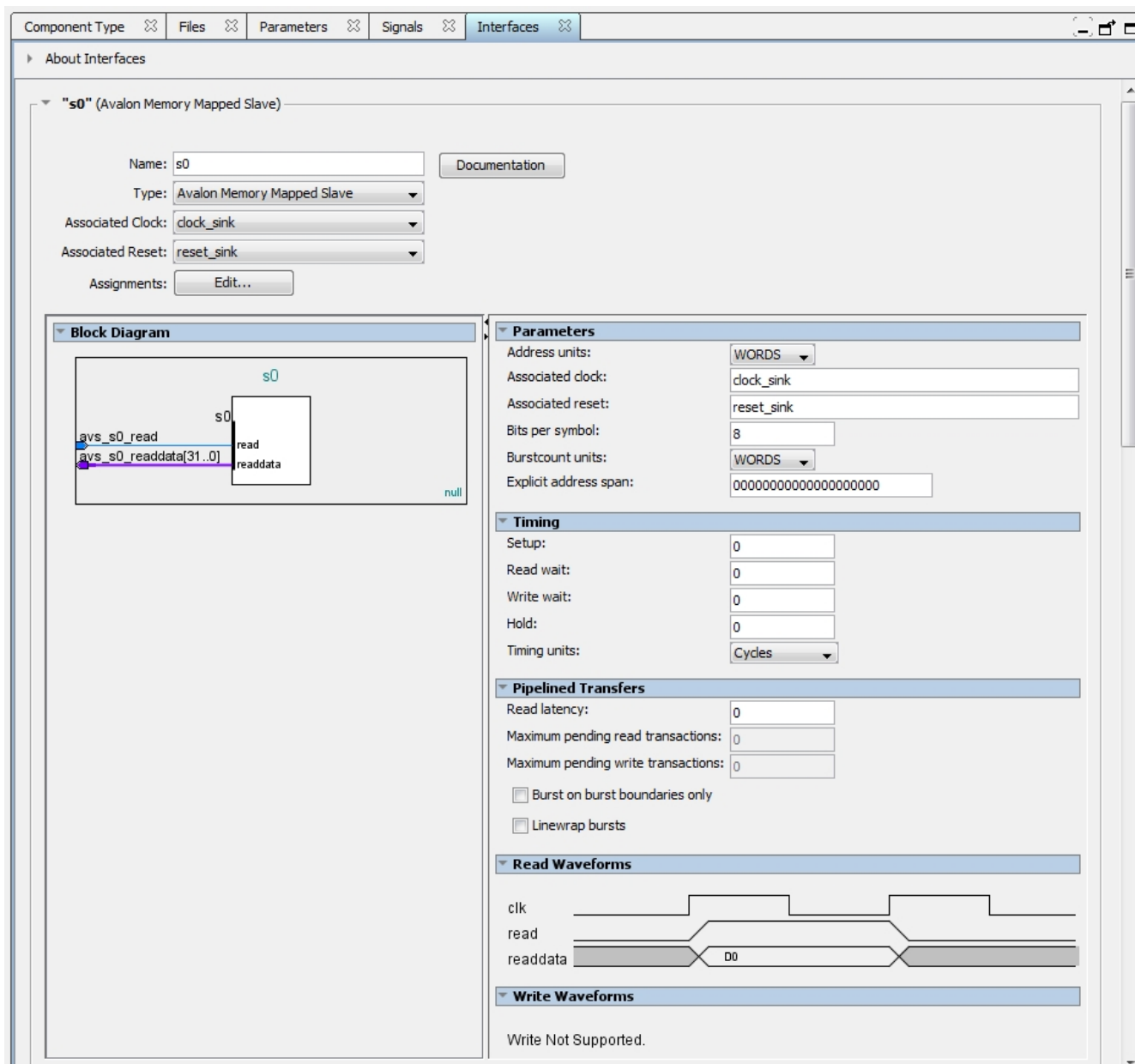


Figure Six: Avalon® Memory-mapped Slave Read Interface

The above screen capture shows the Avalon bus 's0' interface we have created. Of particular interest are the two signals 'avs_s0_read' and 'avs_s0_readdata'. When the Nios II processor is alerted to a pending interrupt and wished to read data from the wrapper, it does so by asserting the read signal, at which time the wrapper must place the data that it is holding from the PERDYN11088 IP onto the 32-bit readdata line. The waveforms shown above illustrate the timing required for this read to occur properly.

Figure Seven illustrates the interrupt signal timing. The wrapper's 'ins_irq0_irq' signal connects to the Nios II processor's interrupt line, and activation causes the Nios II to call its interrupt service routine and read the memory location storing the keypress data, as discussed previously.

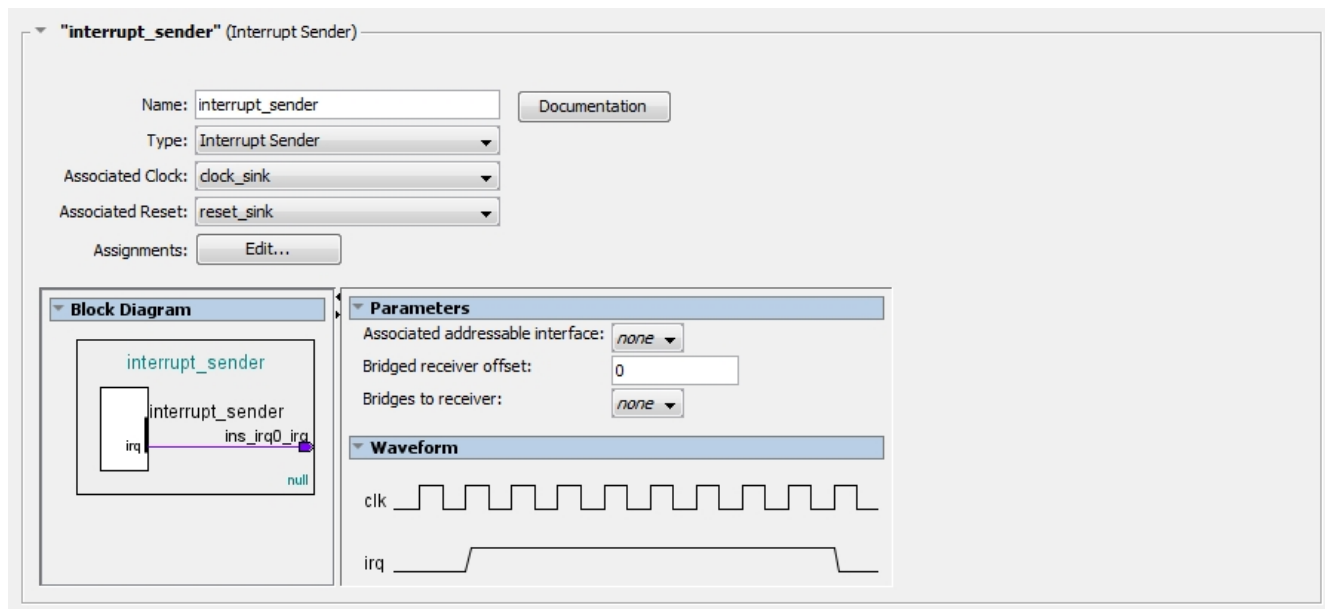


Figure Six: Avalon® Memory-mapped Slave Read Interface

The following three images illustrate the clock, reset, and conduit signals. Notice that the 'conduit' is used to move the row and column keypad connections from the PERDYN11088 IP through the Avalon interface wrapper to the top level pin connections.

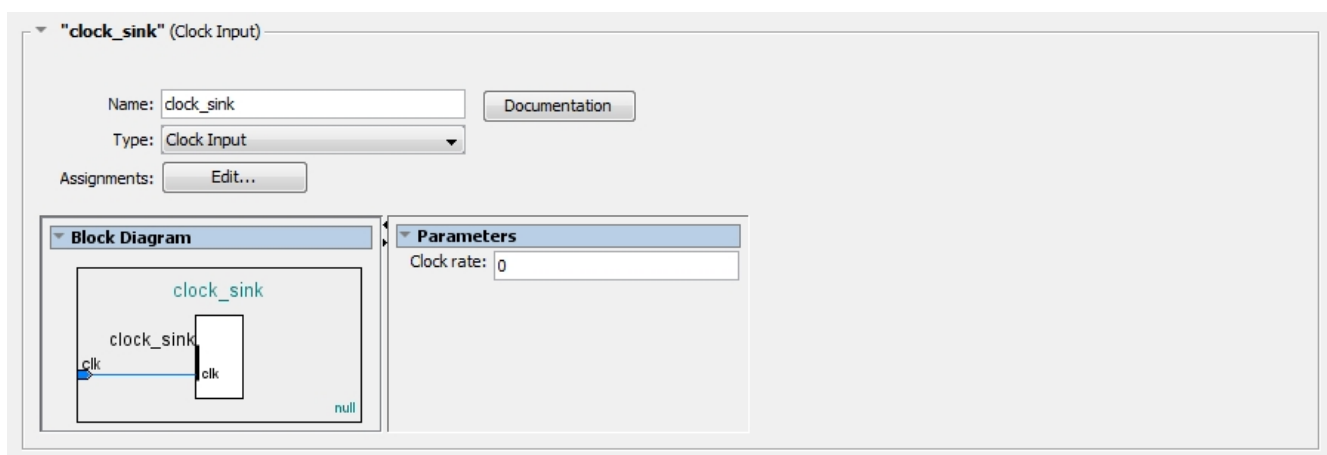


Figure Seven: Wrapper 'Clock' Signal

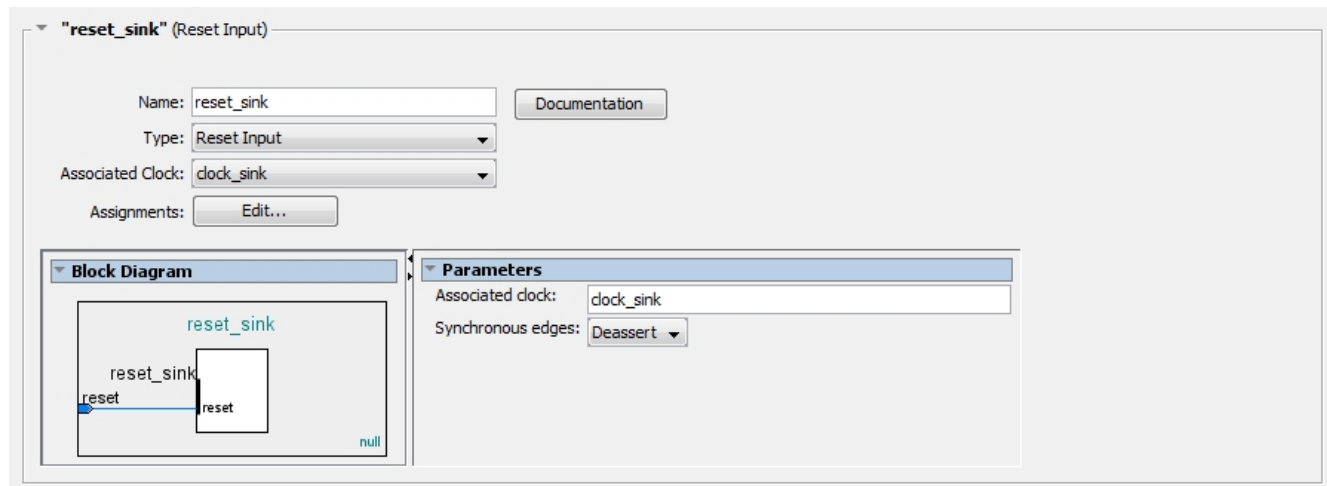


Figure Eight: Wrapper 'Reset' Signal

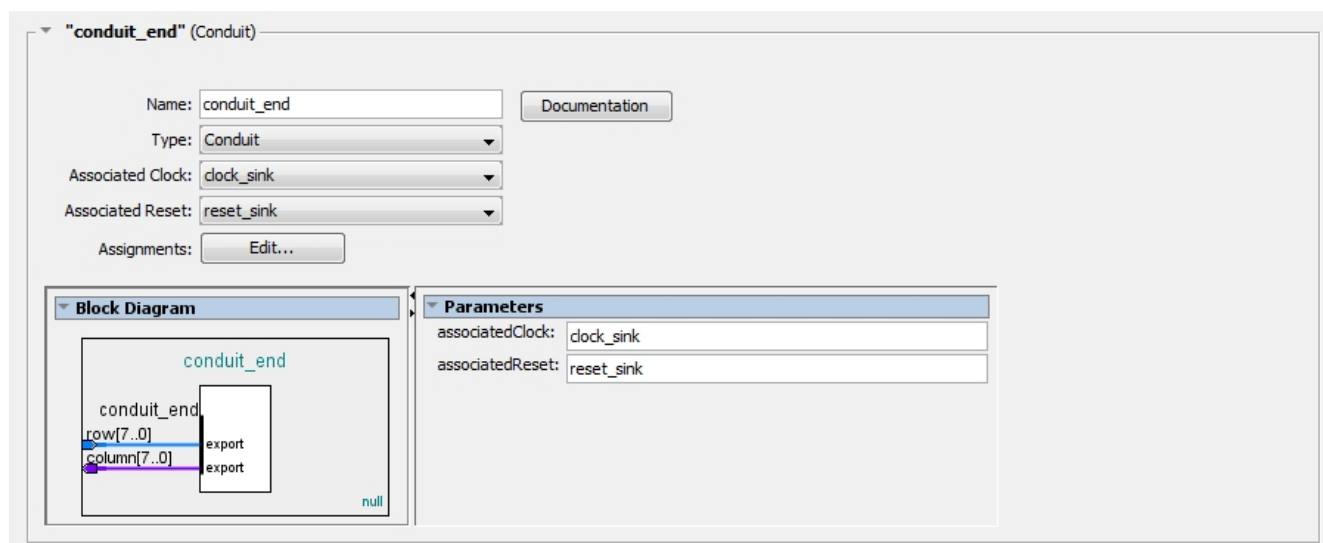


Figure Nine: Wrapper 'Conduit' Signals

If any modifications are made to the wrapper (not necessary for ordinary use), the new component must be created using the Qsys New Component's 'Generate' command, and the newly made component connected to the Nios II processor system as shown in Figure Two.

† Altera®, Nios® II, Quartus® II, and Avalon® are trademarks registered in the US Patent and Trademark Office by Altera Corporation. Verilog® is a trademark registered in the US Patent and Trademark Office by Cadence Design Systems, Inc. PERDYN11088™ and the Peripheral Dynamics logo are trademarks of Peripheral Dynamics, a subsidiary of The Olivet Group Corporation. All other trademarks are property of their respective owners.

APPENDIX A – NIOS II C SOURCE CODE

```

#include <stdio.h>
#include <system.h>
#include <sys/alt_stdio.h>
#include <unistd.h>           // added for usleep
#include <sys/alt_irq.h>     // added for interrupt
#include <io.h>

volatile int readBack;           // value read from an Avalon-MM slave
volatile int printFlag = 0;     // interrupt called, new keypress data available

int pressCount = 0;             // total number of keypresses

static void irqhandler (void*); // function prototype

static void irqhandler(void* context)
{
    readBack = IORD_32DIRECT (PERDYN11088_AVALON_MM_SLAVE_0_BASE,0); //0x3024
    printFlag++;
    usleep(10000);           // hold here 10ms while interrupt signal clears
}

int main()
{
    alt_ic_isr_register(PERDYN11088_AVALON_MM_SLAVE_0_IRQ_INTERRUPT_CONTROLLER_ID,
                       PERDYN11088_AVALON_MM_SLAVE_0_IRQ, irqhandler, NULL, 0x0);
    alt_putstr("PERDYN11088 Keypad Interface IP\n");
    alt_putstr("www.peripheral-dynamics.com\n\n");

    while (1) {
        if (printFlag != 0) {
            alt_printf("(0x%x) Keypress: 0x%x [Interrupts Taken: 0x%x]\n",pressCount++,readBack,printFlag);
            printFlag = 0;
            usleep(50000);
        }
    }

    return 0;                 // should never be reached
}

```