

[Home \(http://blog.embedded.pro/\)](http://blog.embedded.pro/)[Projects \(http://blog.embedded.pro/projects/\)](http://blog.embedded.pro/projects/)[About \(http://blog.embedded.pro/about/\)](http://blog.embedded.pro/about/)

Embedded.Pro

(<http://blog.embedded.pro/>)

Embedded stuff done professionally

Control 7-Segment Display from HPS on DE1-SoC

By Joel Bodenmann (<http://blog.embedded.pro/author/joel-bodenmann/>) · November 2, 2014 · DE1-SoC

(<http://blog.embedded.pro/category/fpga/de1-soc/>) ·  Leave a comment (<http://blog.embedded.pro/7-segment/#respond>)

The DE1-SoC board (<http://blog.embedded.pro/de1-soc/>) is populated with a six digit 7-segment display. All digits are connected to the FPGA. Therefore, in order to control the 7-segment display out of the Linux userspace code, one has to create a new component in QSys that is connected to the AMBA-AXI bus.

But first of all, please note that this is a blog post, not a comprehensive tutorial. The text below does not replace the official Altera documentation. Furthermore, the post does just show some code snippets. However, the fully working project can be found as a download at the very bottom.

QSys Component

I decided to create a component that implements the Avalon-MM bus as it is easy to use and QSys has an implemented translator to attach such a slave to the AMBA-AXI bus. A new component is created by double clicking 'New Component...' in the *IP Catalog* on the left hand side in QSys. After adding some generic component information the required signals to interface the Avalon bus can be added by clicking 'Template -> Add Avalon-MM Simple Slave' in the 'Signals' tab. Furthermore, we need to add an output signal our self to connect the actual 7-segment digits to the component. We do this by adding a signal with the name 'seven_segment', the interface type 'New Conduit...' a width of 42 and the direction set to 'output'. In the 'Signal Type' field we type 'export'.

The output width of 42 is required as the six digit 7-segment display consists of $6 * 7 = 42$ LEDs.

Name	Interface	Signal Type	Width	Direction
avs_s0_address	s0	address	8	input
avs_s0_read	s0	read	1	input
avs_s0_readdata	s0	readdata	32	output
avs_s0_write	s0	write	1	input
avs_s0_writedata	s0	writedata	32	input
avs_s0_waitrequest	s0	waitrequest	1	output
clk	clock	clk	1	input
reset	reset	reset	1	input
seven_segment	conduit_end	export	42	output

seven_segment component signals

In the 'Files' tab we click 'Create Synthesis file From Signals' in the VHDL section to generate a VHDL file. We can now finish the creation of the new component, add it to the system and connect it to the clock, reset and bus lanes. Note that we want to use the 'h2f_lw_axi_master' to connect the new component to the HPS. The 'lw' stands for lightweight.

The next thing we have to do is to export the 42-bit output vector so we can access it from our VHDL code. We do this by double-clicking in the corresponding area and naming it (eg. *output_seven_segment*). The last thing to do is to give the newly added component a base address that is not occupied yet. I chose **0x00040000** in the example below:



7-Segment QSys Component

After generating the VHDL code from QSys we want to wire the actual 7-segment LEDs to the component. The SoC component generated by QSys does now have a new vector (the 42-bit LED vector) in its entity so we can simply map it to our HEX0 to HEX5 vectors in the top level file:

```

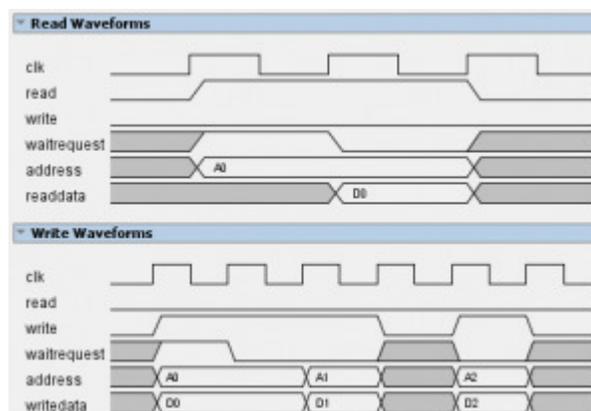
output_seven_segment_export(6+7*0 downto 7*0) => HEX0,
output_seven_segment_export(6+7*1 downto 7*1) => HEX1,
output_seven_segment_export(6+7*2 downto 7*2) => HEX2,
output_seven_segment_export(6+7*3 downto 7*3) => HEX3,
output_seven_segment_export(6+7*4 downto 7*4) => HEX4,
output_seven_segment_export(6+7*5 downto 7*5) => HEX5

```

VHDL Implementation

We have now created a component that is connected to the ARM AMBA-AXI bus. The next step is to implement the actual logic. We have to listen to the Avalon bus and control the LEDs of the 7-segment display.

We can observe the Avalon bus waveforms in the *'Interfaces'* tab of the QSys component properties page:



Avalon-MM Bus Waveforms

Therefore, all we have to do is to listen to the *'write'* signal and handle the *'address'* and *'writedata'* vectors accordingly.

As the Avalon has a bus width of 32-bits, I decided to use the lower 24 bits to transmit the BCD encoded values for all six digits. Therefore, not every digit has its own address but instead all six digits are controlled via the same address. I added another register (= address) to be able to enable or disable (disable as in blank) every digit on its own. Another register was added to control the brightness of the display through a built-in PWM controller.

So from a software developers point of view, the 7-segment peripheral looks like this:

Register	Purpose
0x00	Set display value
0x01	Set display brightness (0 to 255)
0x02	Enable/Disable digit (0 = disabled; 1 = enabled)

The implementation looks like this (note that this is just a code snippet, full code can be found at the bottom):

```
process(all)
begin
    if reset = '1' then

        bcd0 <= (others => '0');
        bcd1 <= (others => '0');
        bcd2 <= (others => '0');
        bcd3 <= (others => '0');
        bcd4 <= (others => '0');
        bcd5 <= (others => '0');
        enable_reg <= (others => '0');

    elsif rising_edge(clk) then

        -- Wait until something interesting happens on the bus
        if avs_s0_write = '1' then

            -- What should we do?
            case avs_s0_address(3 downto 0) is

                -- Write to the display
                when X"0" =>
                    bcd0 <= avs_s0_writedata( 3 downto 0);
                    bcd1 <= avs_s0_writedata( 7 downto 4);
                    bcd2 <= avs_s0_writedata(11 downto 8);
                    bcd3 <= avs_s0_writedata(15 downto 12);
                    bcd4 <= avs_s0_writedata(19 downto 16);
                    bcd5 <= avs_s0_writedata(23 downto 20);

                -- Change the PWM value (0 to 255)
                when X"1" =>
                    pwm_reg <= avs_s0_writedata(7 downto 0);

                -- The enable_reg register
                when X"2" =>
                    enable_reg <= avs_s0_writedata(5 downto 0);

                when others =>

            end case;
        end if;
    end if;
end process;
```

We can clearly see the implementation of the different address mapped features as they are implemented as a simple switch/case statement.

Interfacing from the Linux user space

Assuming that we booted the FPGA configured with the 7-segment controller we are now able to talk to this peripheral from our user space. We use the Linux program **mmap** to map our newly created AMBA-AXI bus member to a virtual memory address.

But first of all we have to run a scrip that reads the QSys file and creates a C-Header containing information about the address space. The script comes with the board examples from Terasic and is called *'generate_hps_qsys_header.sh'*. Note that you have to run this script in the shell that came with the Quartus-II installation that can be opened by executing *Embedded_Command_Shell.bat* in the installation directory. The generated header file contains the following definitions:

```
#define OUTPUT_SEVEN_SEGMENT_COMPONENT_TYPE    seven_segment
#define OUTPUT_SEVEN_SEGMENT_COMPONENT_NAME    output_seven_segment
#define OUTPUT_SEVEN_SEGMENT_BASE             0x40000
#define OUTPUT_SEVEN_SEGMENT_SPAN            1024
#define OUTPUT_SEVEN_SEGMENT_END              0x403ff
```

We can now use these macros in our code to map the 7-segment controller to the virtual memory space:

```
if ((fd = open("/dev/mem", (O_RDWR | O_SYNC))) == -1) {
    printf("ERROR: could not open \"/dev/mem\"...\n");

    return 1;
}

virtual_base = mmap(NULL, HW_REGS_SPAN, (PROT_READ | PROT_WRITE), MAP_SHARED, fd, HW_REGS_BASE);

if (virtual_base == MAP_FAILED) {
    printf("ERROR: mmap() failed...\n");
    close(fd);

    return 1;
}

addr_seven_seg = virtual_base + ((unsigned long)(ALT_LWFPGASLVS_OFST + OUTPUT_SEVEN_SEGMENT_BASE) & (unsigned long)
```

Altera provides the following routine to write to the AMBA-AXI bus slave with their library:

```
alt_write_word(uint32_t addr, uint32_t data);
```

Therefore, we can simply do:

```
alt_write_word(addr_seven_seg, 0x12F4A6);
```

And the number **12F4A6** will appear on the display.

The brightness can now be controlled via this call:

```
alt_write_word(addr_seven_seg+4, 0x150); // Set brightness to 58%
```

An **important** note at this point: As the ARM is four byte aligned, you have to add 4 to reach the next address of the Avalon slave. So slave address *0x02* equals *addr_seven_seg+8*.

The resulting binary can be transferred to the booted Linux running on the HPS using SCP. On execution it should dim the display to a modest value and start counting.

Follow up

I found it extremely useful to not implement the actual logic of a QSys component in the QSys created VHDL file but instead to create a custom VHDL file (I always gave them a *_implementation.vhd* suffix) and simply create an instance of that block in the QSys generated file. This way you don't have to re-generate the QSys code when you changed the implementation of the actual block. This turned out to be a huge time saver during development.

Download

Please use this **on your own risk**: [de1_soc_seven_segment.zip](http://blog.embedded.pro/wp-content/uploads/2014/11/de1_soc_seven_segment.zip) (http://blog.embedded.pro/wp-content/uploads/2014/11/de1_soc_seven_segment.zip)

← [DE1-SoC Project Template](http://blog.embedded.pro/de1-soc/) (<http://blog.embedded.pro/de1-soc/>)

[Using the Serial Port on the DE1-SoC](http://blog.embedded.pro/using-the-serial-port-on-the-de1-soc/) → (<http://blog.embedded.pro/using-the-serial-port-on-the-de1-soc/>)

Leave a Comment

Your email address will not be published.

Recent Posts

- [Improvised: Dummy Load \(http://blog.embedded.pro/improvised-dummy-load/\)](http://blog.embedded.pro/improvised-dummy-load/)
- [DE1-SoC: Using SSH keys \(http://blog.embedded.pro/de1-soc-using-ssh-keys/\)](http://blog.embedded.pro/de1-soc-using-ssh-keys/)
- [DE1-SoC: A better Makefile \(http://blog.embedded.pro/de1-soc-a-better-makefile/\)](http://blog.embedded.pro/de1-soc-a-better-makefile/)
- [STM32F7 I2C HAL \(http://blog.embedded.pro/stm32f7-i2c-hal/\)](http://blog.embedded.pro/stm32f7-i2c-hal/)
- [Custom KiCAD libraries under Windows \(http://blog.embedded.pro/custom-kicad-libraries-under-windows/\)](http://blog.embedded.pro/custom-kicad-libraries-under-windows/)

Categories

- [FPGA \(http://blog.embedded.pro/category/fpga/\)](http://blog.embedded.pro/category/fpga/) (5)
 - [DE1-SoC \(http://blog.embedded.pro/category/fpga/de1-soc/\)](http://blog.embedded.pro/category/fpga/de1-soc/) (5)
- [Improvised \(http://blog.embedded.pro/category/improvised/\)](http://blog.embedded.pro/category/improvised/) (1)
- [KiCAD \(http://blog.embedded.pro/category/kicad/\)](http://blog.embedded.pro/category/kicad/) (1)
- [Laboratory \(http://blog.embedded.pro/category/laboratory/\)](http://blog.embedded.pro/category/laboratory/) (3)
- [Operating Systems \(http://blog.embedded.pro/category/operating-systems/\)](http://blog.embedded.pro/category/operating-systems/) (3)
 - [Linux \(http://blog.embedded.pro/category/operating-systems/linux/\)](http://blog.embedded.pro/category/operating-systems/linux/) (1)
 - [Windows \(http://blog.embedded.pro/category/operating-systems/windows/\)](http://blog.embedded.pro/category/operating-systems/windows/) (2)
- [Qt \(http://blog.embedded.pro/category/qt/\)](http://blog.embedded.pro/category/qt/) (1)
- [STM32 \(http://blog.embedded.pro/category/stm32/\)](http://blog.embedded.pro/category/stm32/) (1)

Archives

- [January 2016 \(http://blog.embedded.pro/2016/01/\)](http://blog.embedded.pro/2016/01/)
- [August 2015 \(http://blog.embedded.pro/2015/08/\)](http://blog.embedded.pro/2015/08/)

- July 2015 (<http://blog.embedded.pro/2015/07/>)
 - June 2015 (<http://blog.embedded.pro/2015/06/>)
 - April 2015 (<http://blog.embedded.pro/2015/04/>)
 - March 2015 (<http://blog.embedded.pro/2015/03/>)
 - November 2014 (<http://blog.embedded.pro/2014/11/>)
-

© 2016 - Embedded.Pro *Designed on rtPanel WordPress Theme Framework (<https://rtcamp.com/rtpanel/>)*.