**Octosynth: A PWM Sound Synthesizer with I2C controller and Custom CPU**

Steve Corey, Nick Elliott, Marko Ljubičić, Jim Squire

An eight-oscillator sound synthesizer was implemented on a chip. Each oscillator is a PWM signal generator, and the PWM duty cycle controls the analog level of the oscillator. An I2C controller sends signals to external analog filters to smooth the PWM outputs into a sound signal. An on-chip CPU with a small ISA coordinates communication between the PWM outputs, I2C controller and off-chip RAM. Different sound effects can be generated algorithmically in software running on the CPU or through look-up-tables stored in RAM. The design of the chip was accomplished with Verilog and prepared for fabrication with Cadence design tools targeting a 180nm CMOS process.

The primary motivation for the project was to learn how to bring a VLSI design to tapeout. The group discussed a few different designs and decided on a sound synthesizer due to its modularity and an ineffable fun factor. We wanted a final product that would be useful, and a synth with a general purpose CPU would provide room for exploration of the limits of the final product. PWM sound generation creates sound that is gritty, raw, and harmonically rich. Musical genres like Industrial EDM or Glitchcore utilize such sounds and are prime targets of the Octosynth. Figure 1 shows a block diagram of the Octosynth.

A CPU provides an interface between an external memory device and several PWM outputs. The CPU is a single core, RISC-based design, with a simple instruction set. A simpler control interface was possible in the form of a state machine with limited PWM accessibility, but a CPU enabled greater creative flexibility. The CPU allows programmatic access to an I2C bus and the PWM outputs through registers specified in the ISA. I2C read and write commands and PWM duty cycles are issued by modifying these registers. Figure 5 shows the ISA.

The PWM modules are controlled through a 16-bit register. The lower 8 bits of the register specify the duty cycle of the PWM signal and at our 10 MHz system clock provide a PWM frequency of 39 kHz. If a different frequency is desired, the upper 8 bits of the register control the counter that toggles the PWM output. With the exception of a value of zero, the value in the upper 8 bits is the value that is counted up to, thus increasing the PWM frequency. An increased PWM frequency results in a loss of the number of used bits in the PWM duty cycle. The PWM register layout is shown in figure 2.

The Octosynth includes an I2C module that serves as a master device for an I2C communications bus. The I2C module is intended to configure digital potentiometers which are part of signal filters to shape the outputs of the PWMs. However, to maximize versatility, the I2C will communicate with any compatible (all 7-bit addressed) devices. When the Octosynth is provided a 10 MHZ clock, the I2C module communicates at the I2C standard 100 Kbit/s rate. The I2C module will also recognize and handle slave clock stretching.

The I2C module is intended to operate with minimal control from the CPU. We accomplished this by dedicating two control registers for the CPU to initialize, while the I2C module idles until an I2C Active bit is set. Inside the control registers, the CPU provides the intended slave device address as well as a R/Wbit, the slave internal register address, and the write data if applicable. Once initialized, the I2C module will complete the read or write operation and when done will clear the start bit as well as send an error status bit if any communications to slave devices were not acknowledged. The CPU is required to poll this start bit to determine when the I2C is free for another operation. The I2C register layout is shown in Fig. 2.

The programmer would load the slave device address and R/W bit into register six using the LI (load immediate) instruction, the internal register device into register seven also using the LI instruction, and if writing would load the write data into register seven using a LUI (Load Upper Immediate) instruction.  Once these are configured, the I2C will begin operation once the I2C Active bit is set by loading a 1 into register six with a Load Upper Immediate instruction.  Once started, the values

should not be changed until the I2C Active bit is cleared by the I2C module.

Testing the final chip design was oriented to unit testing at the component level and integrated testing of the chip as a whole.  Additionally, some components with limited domains of inputs and range of outputs could be comprehensively tested. Components such as the CPU were tested with various sets of assembly instructions as comprehensive testing went beyond the scope of the project.  Testing tools included Xilinx ISE in initial phases of design and then verified further in ModelSim. Fig. 3 shows an example of the ModelSim waveforms that were verified.

Unit testing of the PWM modules involved writing data into the integrated control registers.    Each value written to the control register of each respective PWM module was then checked for the appropriate duty cycle response on the output.  The I2C module was tested for appropriate ACK responses, clock stretching, and read and write operations with appropriate inputs and outputs. To further verify correct functionality, the correct transmission of different sets of data was performed while also introducing scenarios such as clock stretching and varying clock speeds.  The CPU core with an integrated register file was tested using assembly instructions initialized in a Verilog testing environment which instantiated a small memory module. This allowed the CPU to be tested for several hard-coded sets of assembly instructions within the testing environment module. Each testing scenario attempted to drive all ISA instruction types.  A few state machine issues were discovered between decode and execute stages while observing expected outputs from given input instructions. These errors were resolved in ModelSim before moving on to integration level testing.

Integration level testing involved testing Verilog files before pre-defined pad and power modules necessary for VLSI fabrication were introduced.  This step prevented time consuming re-work after fabrication steps were taken. Finally, Verilog modules representing VLSI blocks were introduced and tested in ModelSim to verify functionality, timing constraints and other specifications required by Design Rule Checking and Layout Versus Schematic stages.

Fig. 4 shows the final layout of the Octosynth. Area is 3.46 mm2 including pads, power is 0.84mW. With our target clock of 10 MHz, there is a slack 90.6ns.

The Octosynth provides programmatic PWM signal generation. An on-chip CPU with a small ISA coordinates communication between the PWM outputs, I2C controller and off-chip RAM. Different sound effects can be generated algorithmically in software running on the CPU or through look-up-tables stored in RAM. The chip provides flexible opportunities for unique sound generation. RTL testbenches have confirmed that chip design meets the expected functionality. Area, power, and timing constraints were within acceptable parameters.

*Acknowledgments:*

We would like to thank Edouard Giacomin who created the cell library that enabled our design to finally be completed.

*Bibliography:*

S. Brown and Z. Vranesic, *Fundamentals of digital logic with Verilog design*. Boston: McGraw-Hill Higher Education, 2010.

N. Weste and D. Harris, *CMOS VLSI design*, 4th ed. Boston: Addison-Wesley, 2011.

|  |  |
| --- | --- |
| https://lh4.googleusercontent.com/QSqnMQAR4LPCzY1exMQtNx8puQN4Zc_1ph7Fd4nUc9D0IFTGvUI6NedITNQ10YY6t1n9hSHYXVq8UsVO41KN-EZ7XlM8OEhyAnn5IrpqJyc4BXRvEZwLGCYQxSCG3Fbto7Ijk-o0-gM  **Figure 1: Octosynth block diagram** | https://lh6.googleusercontent.com/ZdxVNERw01zlHeLz49dXOoHUUYMDAWu5mD9jVRuwqBnFIndAF3xnhf3JkZQJiWI7wmLA4zaewXO9nHZnRzvK0C5EyCyLLU7Ora_UQJv9-tiI4GliFMW7EeX8Nt58MDmP_2spv4dK  https://lh5.googleusercontent.com/4G4bKUTKC-PYB1vJnEN8k-PaLnm2i-JFBK_RKyuBw7c4TDOL0jw_AilgQ9_-GrptjixXbK8Td6bm-X2BpB5-WnZT6Lh7n4BytoVKtu1qxR_op82hhwfadze2GMebBFrnCfEJosx4  **Figure 2: Bitwise layout of the** I2C and **PWM registers** |
| https://lh4.googleusercontent.com/E06srgQIRXwQJcNWmTTuUTVm53UTyBQcD2Y5-DZFxOCSvMxC5mzd5jXbex43TC0VAvRWo-1hS_eX0X-RCh02no6tjmcbFJuVyFvdttJKCkyAlP08XP0HDECGDLx3iI2vxXNPDHJNPEU  **Figure 3: Representative example of testing waveforms.** | https://lh5.googleusercontent.com/FRkOKZfYyZ9LiFLErJ772sdUGnHJuAZVTNquiSiCTPLnga3yxnhnKP9miR7adKZqc2XK_d1KS5UO_5xmUIer3Wtl0ZkSDZBC3gzTaIE5DkpjQzWd2tbMcbsSpSM0N-hpMdXQCX3Z2iY  **Figure 4 : Image of the layout of Octosynth.** |
| |  |  |  | | --- | --- | --- | | **OpCode** | **Op Name** | **Layout** | | **0000** | **lw** | reg1(address), reg2(offset), reg(data from mem) | | **0001** | **sw** | reg1(address), reg2(offset), reg(data to mem) | | **0010** | **jump** | COND, CB, FLAG(3bit), JAL, XX, reg (jump destination) | | **0011** | **branch** | CB, FLAG(3bit), 8-bit immediate | | **0100** | **li** | reg1(destination), 8-bit immediate | | **0101** | **oui (or upper immediate)** | reg1(destination), 8-bit immediate | | **0110** | **and** | reg1 ( destination), reg2, reg3 | | **0111** | **or** | reg1 ( destination), reg2, reg3 | | **1000** | **shiftl** | reg1(destination), reg2, 4-bit shift amount | | **1001** | **shiftr** | reg1(destination), reg2, 4-bit shift amount | | **1010** | **add** | reg1 ( destination), reg2, reg3 | | **1011** | **sub** | reg1 ( destination), reg2, reg3 | | **1100** | **lessthan** | XXXX, reg1, reg2 | | **1101** | **equal** | XXXX, reg1, reg2 | | **1110** | **not** | reg1(destination), reg2, XXXX |   **Figure 5: Instruction Set** |  |

|  |  |
| --- | --- |
| * 2018 IEEE International Solid-State Circuits Conference   978-1-5090-4940-0/18/$31.00 ©2018 IEEE  **ISSCC 2018 PAPER CONTINUATIONS** |  |
|  |  |
|  |  |