TITLE: **DMA driven video, ADC and DAC on PIC32: Real time instrumentation on microcontroller**

AUTHORS: Syed Tahmid Mahbub and Bruce R Land

Real time data acquisition systems often need to move a lot of data quickly. The PIC32 microcontroller makes moving data easy because it has four, very flexible, direct memory access (DMA) channels. The DMA frees the CPU to do computational work, while maintaining high data rates. Additionally, the powerful peripherals of the PIC32 allow As an example, we built a device which samples the onboard ADC at 900,000 samples/second, produces a NTSC video signal at 5 megabits/second, feeds a 8 megabit/second SPI DAC (500,000 samples/sec), and still only uses about 8% of the CPU.

PIC32 programming environment

All programming was done using the MPLABX/XC32 environment from Microchip. It is essential to read the PIC32MX Peripheral Reference Manuals [4-11] and the associated C32 Peripheral Library Guide [1]. All coding was done using the register abstractions found in the Peripheral Libraries description. For instance, setting up Timer 2 to interrupt once per video line was done using macros from the library:

```
OpenTimer2(T2_ON | T2_SOURCE_INT | T2_PS_1_1, line_cycles);
ConfigIntTimer2(T2_INT_ON | T2_INT_PRIOR_2);
mT2ClearIntFlag();
```

The first statement specifies: turn on the timer; use the internal clock for a time base; set the timer prescalar to 1:1; set the time-out to line_cycles. The second specifies: turn the interrupt on and set the priority to two. The third just clears the interrupt flag.

One aspect of the PIC32 which makes it very flexible is the ability to remap peripheral units to different physical I/O pins. This is called Peripheral Pin Select (PPS). Groups of eight peripherals are muxed to different I/O pins. For example, the SPI Master Out Slave In/ Serial Data Out (SDO1) signal is muxed in PPS I/O group 2. Mapping the signal to port pin A1, physical pin 2 on the PDIP package, is done by:

```
PPSOutput(2, RPA1, SDO1);
```

Once the pin is attached the peripheral may be opened using

```
SpiChnOpen(spiChn, SPI_OPEN_ON | SPI_OPEN_MODE32 | SPI_OPEN_MSTEN ,
spiClkDiv);
```

Which says to open channel `spiChn` in 32 bit mode, as a master, with clock divider `spiClkDiv`. This command automatically configures the I/O directions for the pins associated with the SPI Module: SDO, SDI and SS. When setting up a PPS assignment, we found it necessary to cycle between three sections in the device datasheet [4]:

(1) Table 11-1 which gives PPS input pin mapping, and Table 11.2 which gives PPS output pin mapping.

(2) The pinout of the actual device we were using.

(3) Table 1.1 which connects the logical pin names to the physical pin numbers for each package.

PIC32 DMA channels

Direct Memory Access (DMA) is a scheme which uses a memory address generator and logic separate from the CPU to access memory. For instance, a DMA channel can move data from the ADC to data memory without CPU intervention. Experiments show that you can push at least four megabytes/sec through the PIC32 DMA subsystem and not affect CPU execution. The application we described here is just under the limit. The PIC32 DMA channels have a number of features. The most interesting is that any hardware event (interrupt) can trigger a DMA transfer, with no associated ISR. The DMA module maintains its own flags for detecting interrupt requests for data transfer start/abort requests. This is completely independent of the interrupt controller. Transfers can be of fixed length, or stop on a specific bit pattern match in the data. Transfers can be from any memory location to any other, which includes every memory mapped peripheral, and can be one-shot or repeating. The channel handles memory wrapping so that one address, for instance the ADC data register, can fill an entire array. The maximum source size, destination size and cell size are all 65,535 bytes which means that up to that many bytes can be transferred on an event. The DMA subsystem can also generate interrupts to signal DMA events, such as completion or transfer abort.

The source and destination addresses are physical addresses, not virtual addresses used by the MIPS core. The reference manual mentions the easy translation between physical and virtual addresses: (Physical address) = (Virtual address) & 0x1FFFFFFF. The PLIB function `DmaChnSetTxfer()` takes in the virtual addresses of the source and destination and does the required virtual-to-physical address conversion as opposed to it being required manually when doing register level operations.

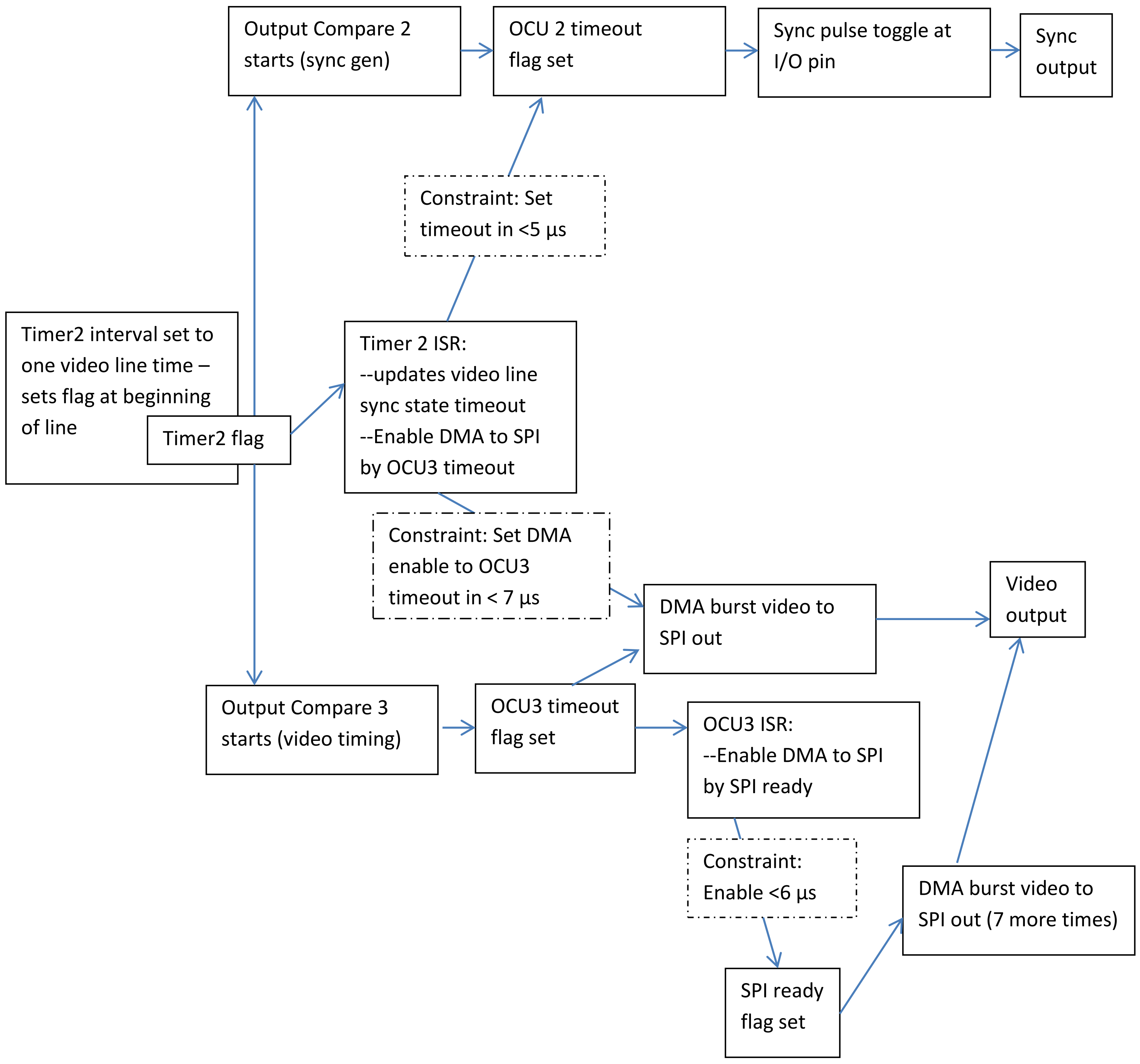
Video generator

We tend to use NTSC video monitors for graphics because they are cheap, rugged, and easy to connect. They are, however, data hungry, requiring constant refresh. The flexible hardware event trigger modes of the DMA make it possible to build a NTSC video driver which uses only a few percent of the CPU. We based the NTSC driver on the approach explained in *Programming 32-bit Microcontrollers in C: Exploring the PIC32* by Lucio Di Jasio, chapter 9 [2], but with more flexibility in video line timing and our own video content drawing utilities. Since video needs accurate timing at the nanosecond scale, it is important that software is never required for critical timing events. All video timing is performed by hardware, with small, interrupt-driven, state machines executing in non-time-critical gaps. Figure 1 shows the hardware/software required. Everything starts when timer 2 times out to signal the start of a new video line. The ISR flag triggers three events: (1) It starts output compare unit 2 to generate a sync pulse on every line (2) It starts output compare unit 3 (OCU3) which starts actual video content seven microseconds later. (3) It triggers an ISR which updates the sync generator state machine and enables a DMA burst when OCU3 times out. The OCU2 sync generator needs input from the ISR before 5 microseconds passes to determine pulse length. The DMA enable must happen within 7 microseconds. The ISR takes around a microsecond to execute. OCU2 then times out and produces a sync pulse at an I/O pin. When OCU3 times out, a DMA burst to the SPI video data port starts, and the OCU3 ISR is triggered to convert the DMA trigger to the SPI done flag. There is a 6 microsecond window to do this and the ISR takes about 500 nanoseconds. The SPI done flag then causes the remainder of the video line to be drawn.

On top of the video data generator we wrote point, line and text drawing routines to fill the display memory and raster (256 wide x 200 tall) with useful graphics. The line drawing algorithm is a standard Bresenham scheme from David Rodgers, *Procedural Elements of Computer Graphics*, 1985 [12]. The text generator uses a bit font identical to the standard 5x7 LCD display font. Figure 2 shows the simple, two-bit, video DAC used. Figure 3 shows some video output.

In the program, main initializes the timers, DMA channels, and output compare units, draws a few strings and lines, then falls into a loop. The loop does a primitive trigger function by waiting until the input ADC conversions pass through zero in the positive direction, then enable the ADC DMA channel, and wait for a screen full of samples. The samples are drawn and the loop starts again.

Fig. 1: The logical flow of the video generator. Note that all time-critical events are hardware triggered.



Output Compare 2 starts (sync gen)
OCU 2 timeout flag set
Sync pulse toggle at I/O pin
Sync output
Constraint: Set timeout in <5 µs
Timer2 interval set to one video line time – sets flag at beginning of line
Timer2 flag
Timer 2 ISR: --updates video line sync state timeout --Enable DMA to SPI by OCU3 timeout
Constraint: Set DMA enable to OCU3 timeout in < 7 µs
DMA burst video to SPI out
Video output
Output Compare 3 starts (video timing)
OCU3 timeout flag set
OCU3 ISR: --Enable DMA to SPI by SPI ready
Constraint: Enable <6 µs
DMA burst video to SPI out (7 more times)
SPI ready flag set

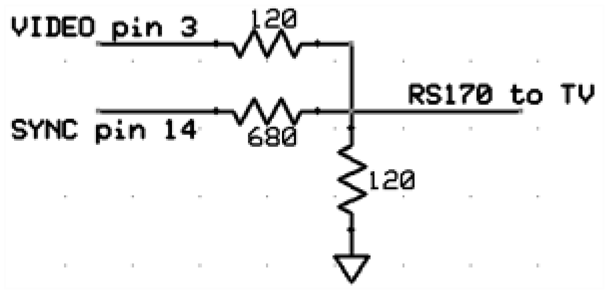


Fig. 2: Video DAC for two-level video (from Di Jasio)

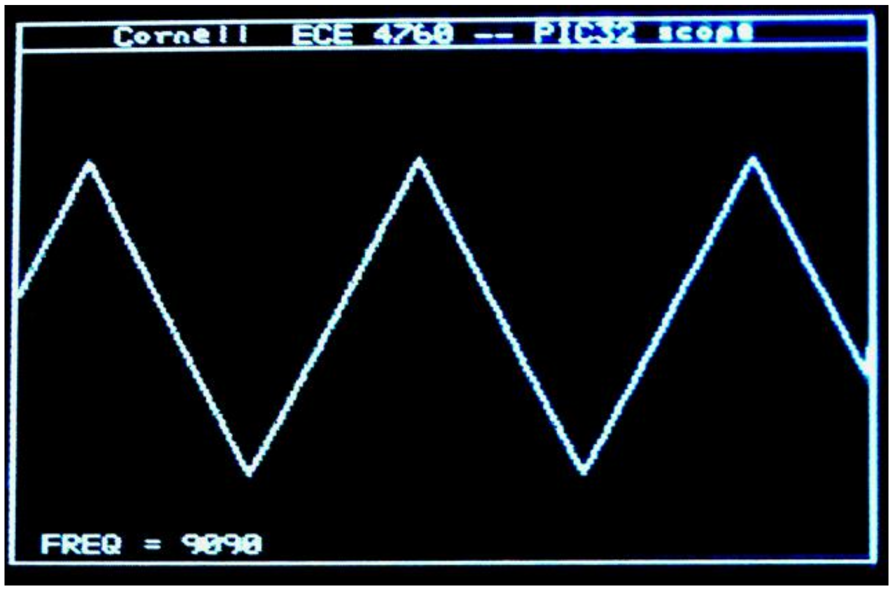


Fig. 3: Video output of the oscilloscope application

Real time ADC

The DMA systems on a PIC32 can move data from the ADC using the ADC done interrupt flag, but without wasting time in an ISR. The ADC can be triggered by a timer period (compare) match, again without software intervention. Doing this means that you can blast ADC into memory at about 900 kilosamples/sec with NO software overhead (except for initial setup)!

The microcontroller has a 10-bit ADC with a bewildering set of options, but we chose a simple mode with options such as channel scanning, alternate buffer and offset calibration disabled. It was set to use AVDD and AVSS as the +V and –V voltage references. The analog input selected was AN11, which is RB13. The ADC was triggered off the Timer 3 period match which would end sampling and begin the analog to digital conversion. No Timer 3 ISR was required. The 30 MHz peripheral bus clock was used as the ADC's clock. This allows the use of an ADC conversion clock period TAD to be 66.7ns which is slightly higher than the minimum specified 65ns. Thus, running the ADC at 30MHz allows the maximizing the ADC clock speed. If the peripheral bus clock was 40MHz instead, the minimum valid ADC conversion clock period is 100ns.

Running at top speed, the ADC can perform about 900 kilosamples per second from one input. The DMA channel was configured to move data from the ADC data register to an array in memory, triggered by the ADC done interrupt, again, with no ISR. The ADC output can be stored in multiple formats. We chose to store it as 16-bit (unsigned) integer. The DMA module transfers 2 bytes (16-bit ADC result) per transaction. The DMA burst terminates after 256 samples. The display code polls the DMA channel-done bit to find out when all the data is available to display.

The ADC and DMA setup were performed using peripheral library functions:

```
SetChanADC10(ADC_CH0_NEG_SAMPLEA_NVREF | ADC_CH0_POS_SAMPLEA_AN11);
// configure to sample AN11

OpenADC10(PARAM1, PARAM2, PARAM3, PARAM4, PARAM5);
// configure ADC using the parameters defined above

#define PARAM1  ADC_FORMAT_INTG16 | ADC_CLK_TMR | ADC_AUTO_SAMPLING_ON
//ADC_CLK_TMR ADC_CLK_AUTO

#define PARAM2  ADC_VREF_AVDD_AVSS | ADC_OFFSET_CAL_DISABLE | ADC_SCAN_OFF |
ADC_SAMPLES_PER_INT_1 | ADC_ALT_BUF_OFF | ADC_ALT_INPUT_OFF

#define PARAM3 ADC_CONV_CLK_PB | ADC_SAMPLE_TIME_5 | ADC_CONV_CLK_Tcy2
//ADC_SAMPLE_TIME_15| ADC_CONV_CLK_Tcy2

#define PARAM4    ENABLE_AN11_ANA
#define PARAM5    SKIP_SCAN_ALL

DmaChnOpen(DMAchan0, DMApri0, DMA_OPEN_DEFAULT);
// channel, channel priority, mode

DmaChnSetTxfer(DMAchan0, (void*) &ADC1BUF0, (void*) v_in, 2, 512, 2);
// (ch, start virtual addr, dest virtual addr, source size,
// dest size, cell size)
// 256 16-bit integers

DmaChnSetEventControl(DMAchan0, DMA_EV_START_IRQ(_ADC_IRQ));
// channel, trigger IRQ -  Trigger on ADC conversion done
```

DMA+SPI to Use External DAC without CPU Intervention

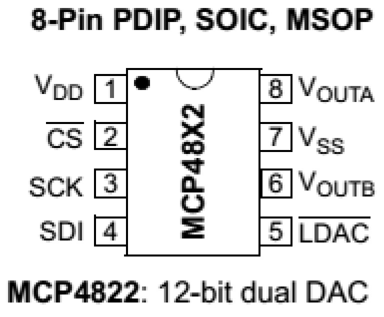


Fig. 4: MCP 4822 pin out (taken from MCP 4822 datasheet [3])

The MCP 4822 is a 2-channel 12-bit DAC with an internal 2.048V reference. It can be supplied a voltage in the range of 2.7V to 5.5V. Since we used 3.3V for the PIC32, we used the same 3.3V supply for the VDD for the MCP 4822. Pin 5 is the active low signal LDAC that is used to synchronize the two DAC channels. When this pin is brought low, the data in the DAC's input register is copied to the output and both outputs are updated at the same time. We just had this tied to ground. VOUTA and VOUTB are the two output pins. The other pins are the regular pins for SPI communication - CS (active low) chip select, SCK serial clock, SDI serial data in.

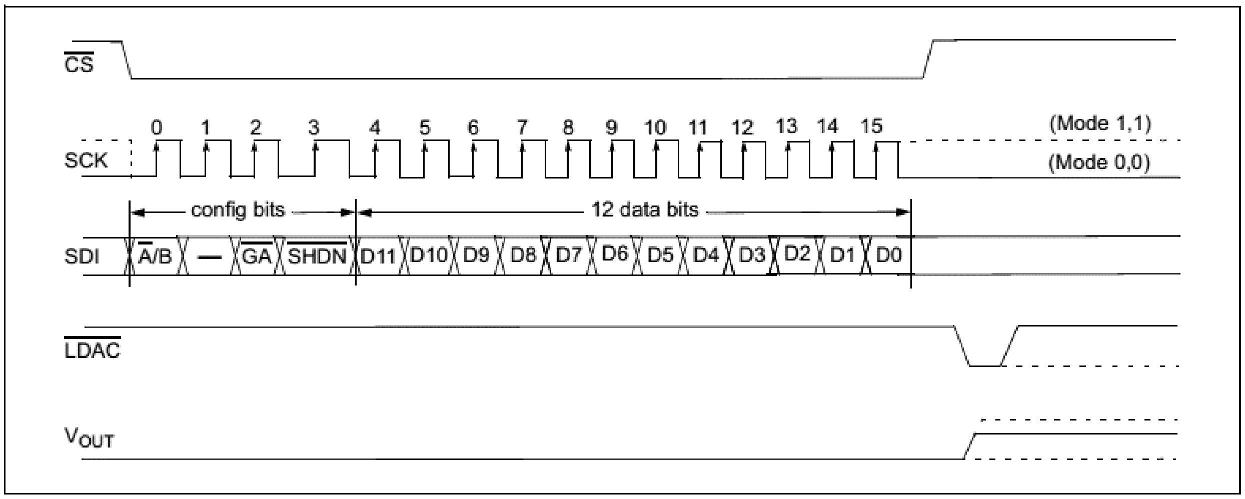


Fig. 5: MCP 4822 Timing Diagram (taken from MCP 4822 datasheet [3])

The MCP 4822 communicates over SPI and is a slave-only device. The write command to the MCP 4822 is a 16-bit command sent over SPI. Bit 15 (A/B) selects which channel data is being sent to: 1 for channel B, 0 for channel A. Bit 13 (GA) selects the gain. When GA = 1, gain = 1; when GA = 0, gain = 2. Bit 12 SHDN is the shutdown signal. When SHDN is low, the output of the DAC is shut down. The 12 data bits from there on: [bit11:bit0] are the 12 data bits for the digital to analog conversion.

In an attempt to make the peripherals and the DAC go as fast as possible, the DAC update frequency was turned up to 500 kHz. This had the advantage of being driven off the same timer as the ADC. While this was beyond spec, the DAC still gave decent output as shown by the output sine wave obtained using an update rate of 737 kHz in a standalone experiment. The update frequency can be easily changed.

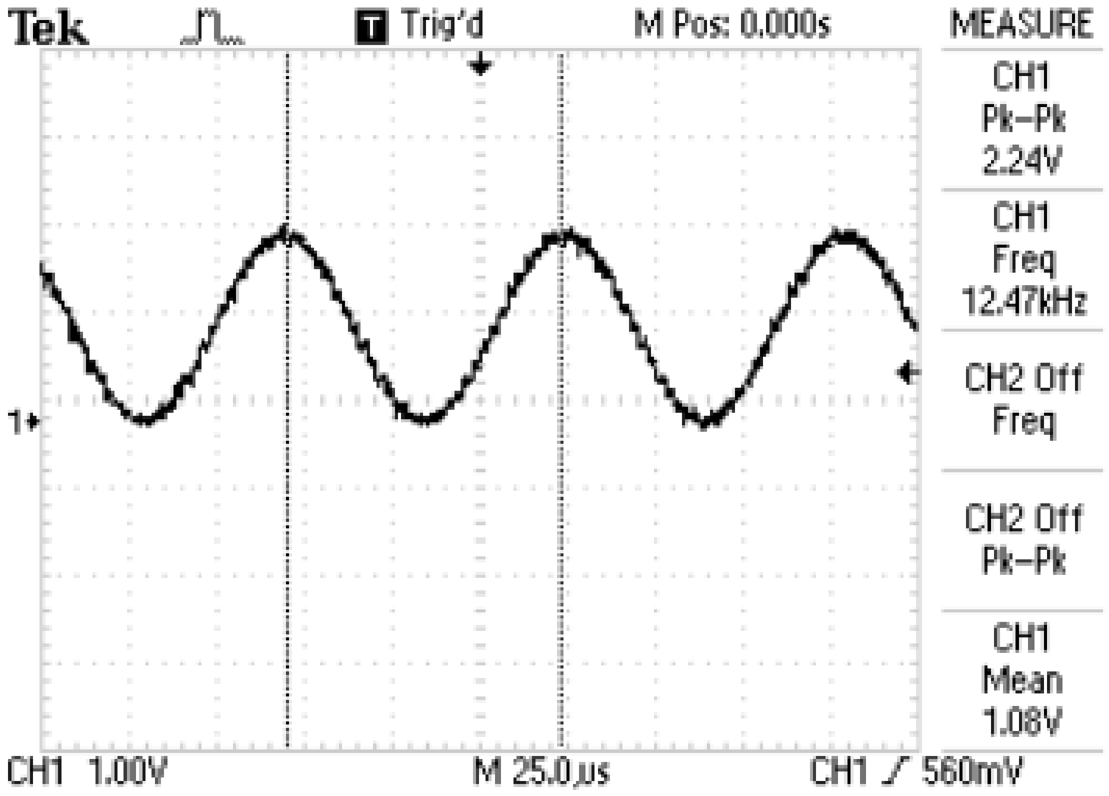


Fig. 6: DAC sample output

The simple way of offloading the SPI update to the DMA module would be to let the DMA channel transfer data to the SPI buffer. The SPI module is configured for 16-bit data transfer (since the MCP 4822 write command requires 2 bytes). This means that the cell size for the DMA channel has been set to 2 (2 bytes to transfer once triggered). The DMA transfer is triggered by a timer interrupt. If we want the entire process to be offloaded from the CPU, the CS (Chip Select) also has to be handled entirely in hardware. For that we used the Output Compare 1 module. To use the OC1 module, either Timer 2 or Timer 3 (or the combined 32-bit timer) has to be used. We used Timer 3 since this was already driving the ADC at 500 kHz.

The idea here was to use the OC module in "dual compare mode continuous output pulses" mode. The OC module in this mode generates continuous pulses: the output pin OC1

- which we used as CS - is set high one PBCLK (peripheral bus clock) after the Timer value reaches OC1R; the OC1 pin is cleared one PBCLK after the Timer value reaches OC1RS. Since CS is active low, we set (PR- 4) [PR=period register] to be OC1RS and a variable CSlength to be OC1R. CSlength was chosen to be 70% of the period register. What this meant was that one PBCLK after the Timer reached the (PR - 4), the OC1 pin went low (CS went low) selecting the DAC. After about 0.70*PR from there, the OC1 pin went high. This means that the CS pin is low for 70% of the period - it goes low a small time right before the DMA transfer happens and is raised high late enough, after the DMA transfer occurs. We determined that 70% of the period (1.4 microseconds) was enough time since that is higher than the time required to shift out 16 bits of data (1.1 microseconds) at 15MHz SPI clock.

The DMA configuration is:

```
DmaChnOpen(0, 3, DMA_OPEN_AUTO);
// (ch, ch priority, mode)

DmaChnSetTxfer(0, source, &SPI1BUF, TABLE_SIZE*2, 2, 2
// assuming 16-bit source entries
// (ch, start virtual addr, dest virtual addr, source size,
// dest size, cell size)

DmaChnSetEventControl(0, DMA_EV_START_IRQ(_TIMER_2_IRQ));
// (ch, trigger irq)

DmaChnEnable(0);
```

The SPI configuration is:

```
SpiChnOpen(1, SPI_OPEN_MSTEN | SPI_OPEN_MODE16 | SPI_OPEN_ON |
      SPI_OPEN_DISSDI | SPI_OPEN_CKE_REV , 2);
```

This enables channel 1 as a master, configured for 16-bit data transmission, disabled SDI pin, configured serial output change from active high (1) to active low (0) as required by the MCP 4822 (see Fig. 5) and set the clock divisor to 2 so that SPI clock = 15 MHz.

Combined Circuit

The combined circuit is a combination of the several different modules mentioned here. It is intended to be a self-contained application to demonstrate the ability to use the PIC32 peripherals in a high data-transfer real-time instrumentation application. The PIC32 drives the DAC with no CPU overhead. It samples this DAC output with the onboard ADC at 500 kHz and transfers the data to an internal buffer, again with no CPU overhead. Then, the corresponding value is displayed on the TV screen in the form of an oscilloscope. The video driver itself does not require much CPU usage. These are made possible by the powerful yet simple DMA modules in the PIC32 as well as the flexible peripherals.

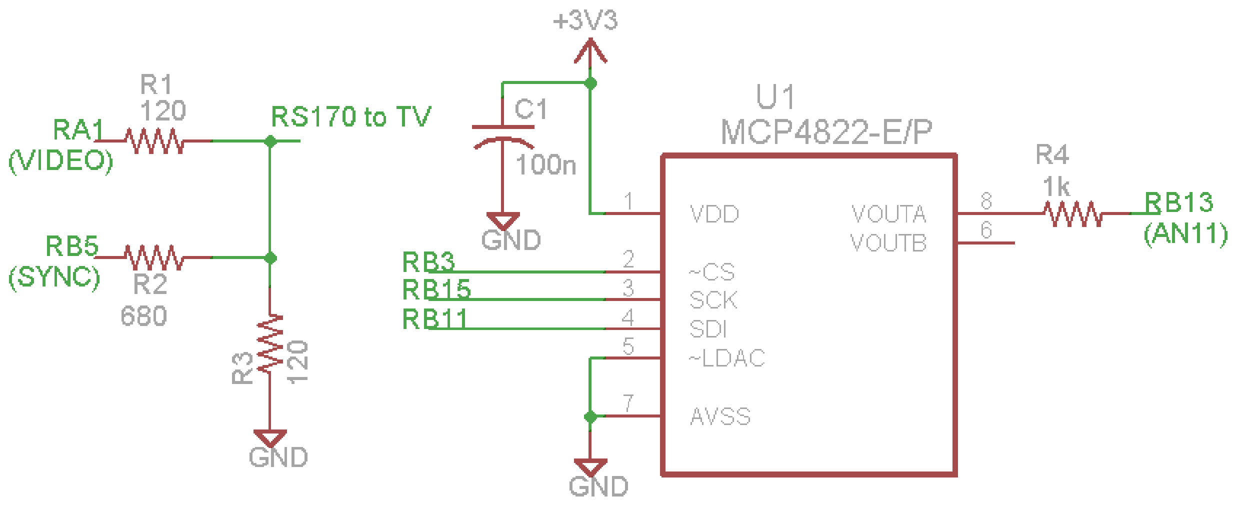


Fig. 7: Complete external circuitry for combination of TV + DAC + ADC

Conclusion

As demonstrated above, the flexible peripherals and the powerful DMA channels of the PIC32 make it possible to use them to create low-cost high-performance systems. With a total cost of under $10, it is possible to create a small lab tool that is capable of generating analog signals, and is capable of sampling inputs to display onto an NTSC television as a small oscilloscope. With only 8% CPU usage, this has plenty of headroom to incorporate other functionality. For example, a simple FFT application may be developed.

There is currently at least one student using the PIC32 in a Masters of Engineering project. The PIC32 will be the microcontroller of choice for the ECE 4760 course (Designing with Microcontrollers) at Cornell starting Fall 2015. We look forward to using it there.

Author Bios

A native of Dhaka, Bangladesh, Syed Tahmid Mahbub is currently an undergraduate student at Cornell University, majoring in Electrical and Computer Engineering. His primary interests in electronics are embedded systems, power systems and analog circuits. Outside of classes, he keeps himself busy with projects involving microcontrollers, robotics and power systems, and writing on his blog. Beyond electronics, he loves watching soccer and cricket. He also plays cricket for the Cornell University club cricket team.

Reference List

[1] Microchip Technology Inc., "PIC32 Peripheral Libraries for MPLAB C32 Compiler". <http://ww1.microchip.com/downloads/en/DeviceDoc/32bitPeripheralLibraryGuide.pdf>

[2] Jasio, Lucio Di. Chapter 9. *Programming 32-bit Microcontrollers in C: Exploring the PIC32*. Elsevier, 2008. Print.

[3] Microchip Technology Inc., "8/10/12-Bit Dual Voltage Output Digital-to-Analog Converter with Internal VREF and SPI Interface", MCP4822 Datasheet. <http://ww1.microchip.com/downloads/en/DeviceDoc/22249A.pdf>

[4] Microchip Technology Inc., "32-bit Microcontrollers (up to 256 KB Flash and 64 KB SRAM) with Audio and Graphics Interfaces, USB, and Advanced Analog", PIC32MX250F128B Datasheet. <http://ww1.microchip.com/downloads/en/DeviceDoc/60001168F.pdf>

[5] Microchip Technology Inc., Reference Manual, "Section 17. 10-bit Analog-to-Digital Converter (ADC)". <http://ww1.microchip.com/downloads/en/DeviceDoc/61104E.pdf>

[6] Microchip Technology Inc., Reference Manual, "Section 31. DMA Controller". <http://ww1.microchip.com/downloads/en/DeviceDoc/60001117H.pdf>

[7] Microchip Technology Inc., Reference Manual, "Section 8. Interrupts". <http://ww1.microchip.com/downloads/en/DeviceDoc/61108G.pdf>

[8] Microchip Technology Inc., Reference Manual, "Section 12. I/O Ports". <http://ww1.microchip.com/downloads/en/DeviceDoc/61120E.pdf>

[9] Microchip Technology Inc., Reference Manual, "Section 16. Output Compare". <http://ww1.microchip.com/downloads/en/DeviceDoc/61111E.pdf>

[10] Microchip Technology Inc., Reference Manual, "Section 23. Serial Peripheral Interface (SPI)". <http://ww1.microchip.com/downloads/en/DeviceDoc/61106G.pdf>

[11] Microchip Technology Inc., Reference Manual, "Section 14. Timers". <http://ww1.microchip.com/downloads/en/DeviceDoc/61105F.pdf>

[12] Rogers, David F. *Procedural Elements for Computer Graphics*. New York: McGraw-Hill, 1985. Print.