Arduino Due Low frequency NMR

May, 2020
Carl Michal

The Due NMR spectrometer provides a sine-wave (or other user-defined) waveform output, sampled at a constant 500,000 sample/sec rate of 12 bit samples. The DAC output range spans roughly Vcc/6 to 5 Vcc/6. (Vcc is nominally 3.3 V). The input is sampled by a 12-bit ADC, also at 500,000 samples per second. Input samples are zero-shifted, then multiplied by in-phase and quadrature values of a digital sinusoidal reference, then filtered, decimated and delivered to the host computer. In normal operation, the output waveform and the demodulator reference frequency are identical, however, the source signal frequency can be set independently if desired.

Pulse programs are downloaded over the native USB interface "on the fly," where a subsequent transient is programmed during the final event of the current transient to ensure that timing of all events is precise. Pulse programs can include flow control instructions such as (nested) looping and subroutines. The number of levels of loops or subroutines is a compile-time option, currently set at 4.

In addition to the transmit and receive, 25 GPIO outputs may be set in synchrony with the pulse outputs, though with higher timing resolution. Transmit and receive events always begin and end on 2 µs boundaries (due to the 500,000 sample/s clock rate of the ADC and DAC), while GPIO output events are timed with a 42 Mhz clock (~ 23.8 ns resolution).

The GPIO pin outputs are set in an interrupt, and the timing has some jitter. There is a (default) compile-time option to have GPIO pin outputs to be set exactly on-time via an external latch chip. If this option is active (which is by default) the GPIO outpus are set immediately after the start of the previous event.

Several ADC parameters can be set, including the setting of the input source multiplexer, single-ended / differential mode, and the input gain can be changed by a factor of 4.

Building the Code

The code is an arduino sketch, built in the arduino IDE. If built as-is in a default arduino installation, it probably **won't** work. The code was developed using gcc-4.8.3-2014q1 (found in `~/.arduino15/packages/arduino/tools/arm-none-eabi-gcc`.
The default compile options use -Os, to optimize for smallest code size. The code runs much more quickly if compiled with -O3. To change the optimization level, find `platform.txt`, somewhere like: `~/.arduino15/packages/arduino/hardware/sam/1.6.12/platform.txt` and change -Os to -O3 (in three places).

One strange compilation issue is that the duration taken to execute the critical interrupt that processes received data samples and prepares new DAC samples depends on factors that are hard to control. This appears to be a code alignment issue. An assembly alignment directive (.align 4) seemed to affect it, but how this directive affects the code has changed as the code evolved. After making changes to the code, the duration of these interrupts should be checked (they can be observed on pin 7 after uncommenting the appropriate FLAG_ON/FLAG_OFF lines). 83 us is the duration currently observed when the code is compiled with -O3, with the default clock speed of 84 MHz.

Communications

All communications to the host computer are over the native USB serial interface. Commands are specified by a single byte, sometimes followed by additional bytes which, if present, carry binary coded (little-endian) integers. For all commands that are followed by additional bytes, with the exception of 'W', and ‘D’ the additional bytes should be appended so that the command byte and the additional bytes are sent with the same write() function call as the command byte. This is so that the additional bytes are transmitted in the same USB packet to the Due and the reception code will never need to wait for an additional packet to arrive. The cases of 'W' and ‘D’ are distinct, and described below.

Commands

The commands that can be sent are:

Q Query – request identification.

Fnnnn Set the frequency of both the signal source and the demodulator. If they were different, they will be set the same. F is followed by a four-byte frequency tuning word (FTW). The nearest FTW to a frequency is: (int32_t) round(frequency x 2^{32}/500000). The actual frequency that will be produced is 500000 Hz x FTW/2^{32}.

f Request return of the demodulator FTW.

Tnnnn Set the signal generator FTW, but leave the demodulator where it was. Format as for F.

t Request return of the signal generator FTW.

U Use the downloaded waveform buffer. [may be removed]

u Use the built-in sine wave. [may be removed]

C Set the channel number for the ADC. Power-up default is AD7 (arduino pin A0).

c Request return of the ADC channel number.

D Download a pulse program. The detailed program format is described below.
The D command is followed by 6 bytes representing 3 16-bit binary numbers. These numbers indicate the number of 32-bit words in each of three pulse program segments: GPIO, transmit, receive. The D command and these 6 bytes should be transmitted and flushed and then followed by the binary pulse program.

S Safe: if a program was running it is terminated. GPIO outputs and the DAC output are placed in their ‘safe’ states.

Y Start execution of the pulse program.

x Request return of status and flag bytes.

Pnn Set the period of the PWM oscillator. The frequency is 1,000,000 Hz/period.

Knnnn Set the frequency of the PWM oscillator clock. This will modify the 1MHz base clock for PWM output. This number is a frequency in Hz, and should be an integral submultiple of the Due clock speed (84,000,000 or 100,000,000).

p Request return of the PWM period.

W<data> Download a waveform into the user waveform buffer. The W is followed by 16384 16-bit numbers. Only 12 bit values from 0-4095 should be used. The W should be transmitted and flushed through the serial buffer before writing the rest of the data (better alignment of the data if it comes in a separate USB packet from the W). The Due will not respond until it has received the entire 32768 byte payload after the W.

Gcv Write a digital pin high or low. 'c' is the arduino pin number, and 'v' is the value to write to it. [likely to be removed]

gc Read a digital GPIO pin (pin c). [likely to be removed]

L Set ADC in differential mode. See table below for which pins are used.

l Set ADC in single-ended mode (power-up default).

O Turn on ADC offset mode. In this case the ADC center value is nominally at Vcc/2, even when the gain is higher than 1. Only matters in single-ended mode.

o Turn off offset mode. Here, in single-ended mode the ADC values span a range from 0-Vcc, or 0-Vcc/2 or 0-Vcc/4.

An Gain. Set ADC gain. n is one byte binary with a value of 0-3.

Return data:

Communications back from the Due **always** come in 64 byte packets. There are two types of packets. The first is a data packet, which has a 4 byte header followed by 60 bytes of sample data. The second type of packet is an information packet, which is sent either in response to a query, or to indicate a change in the state of the external reference synchronization. Information packets are ASCII coded strings. The two packet types can be distinguished by checking the most-significant bit in the first byte of the packet. It is always high for a data packet, always low for an information packet.

Data Packets:
The 4-byte header consists of: A status byte, where the most-significant bit is always set to indicate that the packet is a data packet, the other bits report other status information described below. The status byte is followed by a flag byte, followed by a 16 bit integer that gives the number of data words in the packet. The 60 bytes of sample data are 16 bit integers, of in-phase (I) and quadrature (Q) samples which have been demodulated, filtered, and down-sampled. These are interleaved as I_0Q_0 I_1Q_1 I_2Q_2 ... $I_{15}Q_{15}$

The bits of the status byte are:

STATUS_RECEIVERON 1
indicates if the receiver is currently filtering and downsampling data. Used internally

STATUS_R_RELOADED 2
indicates that a pulse program has been downloaded during the final event of a sequence. Used internally

STATUS_T_RELOADED 4
indicates that a pulse program has been downloaded during the final event of a sequence. Used internally

STATUS_LAST_EVENT 8
indicates that the final event of a sequence is currently executing.

STATUS_SHUTDOWN 16
indicates that the pulse program reached the end of the final event (of the GPIO program), without receiving a new program download.

STATUS_RUNNING 32
indicates that a pulse sequence is running.

STATUS_USE_DOWNLOADED 64
indicates whether the memory buffer for a user defined waveform is being used for dac output.
[likely to change or be removed?]

STATUS_MARKER 128
Always high, to indicate that this is a data packet.

The flag byte contains:
FLAG_GAIN 1 | 2
The first two bits show the value of the ADC gain. The operation of this is a little strange, and depends on whether the ADC input is in single-ended or differential mode. In single-ended mode, gain settings of 0, 1, 2, 3 give actual gains of 1, 1, 2, and 4. In differential mode, the gains are 0.5, 1, 2, and 2.

FLAG_DIFFERENTIAL 4
indicates whether the ADC is in differential mode or single-ended.

FLAG_OFFSET 8
In single-ended mode, turning on the offset keeps the center of the ADC range at VCC/2.

The remaining flag bits behave differently, and indicate the detection of errors. These bits are turned on when the error is detected, and reset when reported. If the error appears in successive packets, the problem has been detected again.

FLAG_INPUT_OVERFLOW 8
An ADC value was found that was close to one of the limits. Only one sample out of every 50 is checked, brief spikes in the input signal may not be caught.

FLAG_OUT_BUFF_OVERRUN 16

The output buffer overflowed and data was lost.

FLAG_DMA_XRUN 32
The DMA buffer for the ADC or DAC suffered an overrun or underrun

<u>Information packets:</u>
These are also always 64 bytes, but the messages are much shorter. The full 64 byte packet is transmitted because it aligns with the USB buffer size on the Due, and the constant packet size allows the receiving program on the host computer to always knows where to look for the marker bit that distinguishes data from information packets. Information packets are ASCII strings. Numerical values here are provided as strings. The possible information packets are:

"LAST EVENT"
- indicates that the final event of the (GPIO) pulse program has started.

"SHUTDOWN"
- indicates that the sequence was stopped because the final event was completed without a new program download.

"Due NMR vx,y,z"
- sent in response to the Q command

"F: %i"
- gives the frequency of the demodulation, as a frequency tuning word. Given in response to the 'f' query.

"T: %i"
- gives the frequency of the signal generator, as a frequency tuning word. Given in response to the 't' query.

"P: %i"
- gives the period of the PWM waveform on D53. Given as a period measured in 1us ticks.

"C: %i"
- gives the channel number that the ADC is set to sample.

"pin: %i val: %i"
- provides the digital value on a GPIO pin, in response to a 'g' command. [likely to be removed]

"status: %i, flags: %i"
- provides the status and flags bytes (as parts of ascii coded strings!) in response to the 'x' command.

"WATCHDOG TRIGGER %i"
- indicates that the watchdog shut down a pulse sequence. There are two possible causes for this: One is that the data overflowed the output buffer because transmitting data over the USB interface had fallen behind. The second possibility is that there was a DMA over- or under-run. These conditions would should not normally happen, but might be triggered by exceptionally complex pulse sequences. The number following the string tells which of these conditions occured. The value is a bitmask, where 1 indicates a data overflow and 2 indicates a DMA over-/under-run. A 3 indicates that both were detected.

Watchdog errors can occur if the host computer does not read data transmitted by the Due quickly enough.

Example Code
Some sample code in python that talks to the lock-in:
`test_nmr.py`

Pins

The ADC has a multiplexer and the input can be chosen using the 'C' command. The current multiplexer setting can be queried with the 'c' command. The pin assignments are:

ADC channel number	Arduino pin	GPIO pin id
0	A7/D61	PA2
1	A6/D60	PA3
2	A5/D59	PA4
3	A4/D58	PA6
4	A3/D57	PA22
5	A2/D56	PA23
6	A1/D55	PA24
7*	A0/D54	PA16
8	D20	PB12
9	D21	PB13
10	A8/D62	PB17
11	A9/D63	PB18
12	A10/D64	PB19
13	A11/D65	PB20
14	D52	PB21

* A0 is the power-up default input

If the ADC is placed in differential mode, channels are combined: 0-1, 2-3, 4-5, etc. The manual says that the channel number should be set to the even numbered channel, but it appears that setting it to either of the two channels works.

Other pins

Function	Arduino pin id	GPIO pin id	function
DAC output	DAC0/D66	PB15	DAC0

Pulses for GPIO latch.	D2	PB25 (periph B)	TIOA0
PWM output	D53	PB14 (periph B)	PWMH2
ADC interrupt active flag (debugging)	D7	PC23	
LED (debugging)	D13	PB27	

Looping D53 to A7 allows testing of the input capture/synchronization – the PWM output is produced somewhat differently from the main signal, so most PWM frequencies can not be exactly reproduced by the signal source.

Looping DAC0 to A0 allows the receiver to see the source signal.

GPIO outputs
PortC of the Arduino is used for GPIO outputs. The pin assignments are:

C.0 = N/A
C.1 = D33
C.2 = D34
C.3 = D35
C.4 = D36
C.5 = D37
C.6 = D38
C.7 = D39
C.8 = D40
C.9 = D41
C.10 = N/A (OPCODE0)
C.11 = N/A (OPCODE1)
C.12 = D51
C.13 = D50
C.14 = D49
C.15 = D48
C.16 = D47
C.17 = D46
C.18 = D45
C.19 = D44
C.20 = N/A (OPCODE2)
C.21 = D9
C.22 = D8
C.23 = D7 // Used as interrupt duration flag.
C.24 = D6
C.25 = D5
C.26 - D4/D87
C.27 =
C.28 = D3
C.29 = D10/D77
C.30 = D72 (not easy – probably unavailable)
C.31 = N/A

Internals

The basic timebase is a 32 bit register that stores the value of “the phase” of the signal and reference. For each DAC/ADC sample, the phase registers are incremented by a frequency tuning word. The sine wave signals for the DAC and reference multiplication are stored in look-up tables. There are two tables, on 32768-sample long table of 16-bit values (from -32767 to +32767) for reference multiplication, and a second 16384-sample long table of 12 bit values (0 to 2047) for the DAC.

The Atmel SAM3X/SAM3A series data sheet is essential for understanding the source code and registers accessed. This is Atmel document 11057.

Filters and Decimation

After multiplication by the reference frequency, the samples are filtered with two stages of Cascaded-integrator-comb (CIC) filters. The first stage is 2-level (CIC2) filter, the second has 5 levels (CIC5). Each filter incorporates a decimation factor of 5. Samples enter the filter at 500,000 samples per second, and exit at 20,000 samples per second.

The CIC filter response is well known, and has the nice feature that it produces nulls where the input spectrum folds onto 0. A nice reference is the AD6620 datasheet. The transfer function for the first stage filter is:

$$H(f)=\left(\frac{\sin(\pi\frac{M_2 f}{f_{samp}})}{\sin(\pi\frac{f}{f_{samp}})}\right)^2$$

where M_2 is the decimation.

Similarly, in the second (CIC5) stage, the response is:

$$H(f)=\left(\frac{\sin(\pi\frac{M_5 f}{f_{samp}})}{\sin(\pi\frac{f}{f_{samp}})}\right)^5$$

. Both M_2 and M_5 are 5. In the first stage, f_{samp} is 500,000, in the second stage, f_{samp} is 100,000. The resulting filter response, on a linear scale, looks very much like a Gaussian having a standard deviation of 5 kHz. On a log scale the response is much more complicated, but it is clear that the filter response nulls at 20, 40, 60, etc kHz.

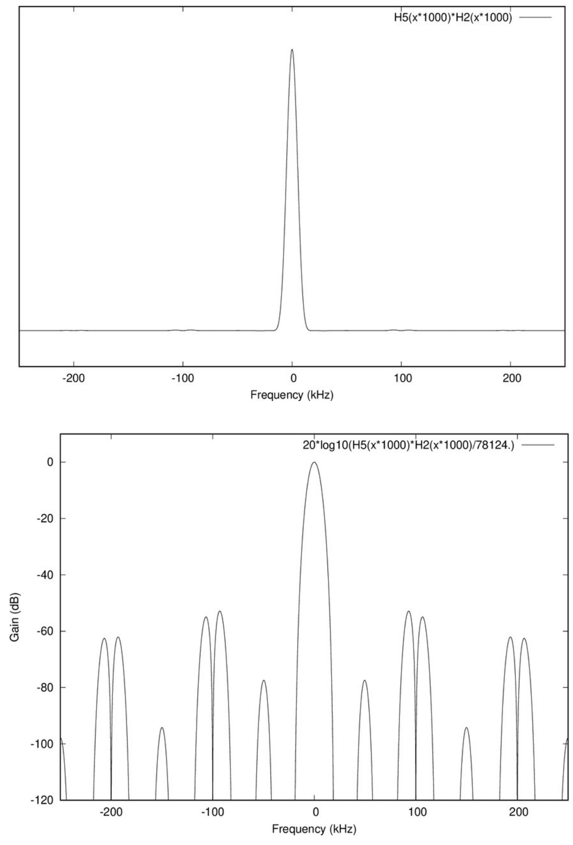

Figure 1: Theoretical CIC filter response for CIC2 + CIC5 with decimation of 5 in each stage and starting sample rate of 500 ksps. Upper panel is a linear scale, lower panel on a log scale.

Rounding Errors

Just before the first stage filtering, after DAC samples are multiplied by sin/cos values, the results of the multiplication are bit shifted where they should be divided. Negative numbers are rounded slightly differently when bit-shifted. Because this rounding occurs 13 bits below the LSB of the DAC sample, this rounding error is not expected to ever be significant.

Timers

Internally, four timer units of the Sam 3X8E are used. The nomenclature of the timers is **very** confusing. There are nine timers organized into three blocks. The symbols TC0, TC1, and TC2, are used in different places to refer to the different blocks of three timers or to the different timers within a block. In the Atmel manual, the Timer chapter mostly talks about a single block and so there TC0 refers to the block of three timers. In that context there is no such thing as TC3. However, in the source code, the interface to the timer registers uses two different conventions. There are constructions like: `TC1->TC_CHANNEL[0].TC_CV`, which refers to the timer value of the 1st channel in the second block. But there are also constructions like: `pmc_enable_periph_clk(ID_TC3);` and `NVIC_EnableIRQ(TC3_IRQ)n;` which refer to the same timer. So TC3 is the same timer as `TC1->TC_CHANNEL[0].`

The code mostly uses the first block of three timers. TC2 is used to trigger the DAC and ADC. It counts at MCK/2 (42 or 50 MHz) for 2us, then triggers the DAC and ADC, then resets and repeats. Both use DMA to load/store 50 samples at a time.

TC0 is used for the pulses at the start of each cycle.

Both of these timers are started at the same time with an internal SYNC start command.

Timer 3: TC3, aka (TC1, channel 0) is used as a watchdog to intervene if the main loop is being starved of cpu time and to check for overruns/underruns of the DMA buffers.

Interrupts

Much of the key processing is handled in interrupts. When the ADC DMA sample buffer is full, an interrupt is triggered. The handler loads the next 50 DAC samples, then does the multiplication, filtering, and decimation of the samples. Two complex samples are loaded into an output buffer for each 50 ADC samples.

A second interrupt handler sets up the pulses that occur at the start of each cycle.

A final interrupt handler is set up as a watchdog to ensure that the pulse and input capture ISR’s don’t starve everything else of CPU time. If it sees that data is getting lost it will shut off pulses and input capture. If it sees a DMA over/underrun, it will shut off the pulses and input capture and will restart the ADC and DAC.

Pulse Program Format

The binary pulse program consists of three segments, first for the GPIO outputs, second for the transmitter and finally for the receiver.

GPIO events consist of two or three 32-bit words. The first word is the duration of the event, measured in 42 MHz clock cycles. The second word contains the output bits, this word is directly written to the SAM3XE port C output, though not all bits are used. The table above of pins describes which bits correspond top which output pins. Bit 23 is currently masked off and is used as a flag to measure

interrupt performance. Three of the unused bits (20, 11, and 10) are used as an opcode. The meanings of the opcodes are:
0: normal event. When over, proceed to the next event.
1: loop start. In this case there is a third word which contains the number of loop iterations, which must be at least 1.
2: loop end.
3: last event. Used to mark the final event. This triggers a message so the host can download a new program on the fly. If this last event completes without a new program download, execution is halted.
4: subroutine. In this case there is a third word which contains the address of the subroutine start. The address is just the word number in the pulse program.
5: return

Transmit events consist of one to three words. The lower 28 bits of the first word is the event duration measured in 2us ADC/DAC ticks. The most significant bit indicates whether the output is turned on or off, the next three bits are used to indicate the opcode. The opcode meanings are almost the same as for the GPIO opcodes, however it is the GPIO last event opcode that triggers the LAST EVENT message to the host.

There is one additional opcode:
6: phase reset. The phase of the output signal is reset to 0 at the start of this event.

If the rf is turned on, then the second event is the phase for the pulse. (1<<31) is 180 degrees, (1<<30) is 90 degrees. All 32 bits can be used.

If the opcode is loop start or subroutine start, the final word contains the number of loop iterations (which must be at least one) or the subroutine event address.

Receive events are similar to transmit events, except the MSB of the duration is used to indicate if the receiver is on, and there is no phase word, all receiver phase shifts must be done by the host. The phase reset opcode is available.

Receive events should be multiples of 50 us long, as that is how long it takes to generate and filter one point.

The receiver filter state is reset during any ADC interrupt where no points are received. To guarantee that this happens, there should be an event that is at least 200us long between receive events.

The last event opcode should be used in the final event for each program segment, though they do not need to appear at the same time. Events in the different segments do not need to align. If the transmit and receive programs are shorter than the GPIO program, the final events are repeated. If the GPIO program is shorter, then the program terminates. Probably there are corner cases where unexpected things can happen if the programs have different lengths in a way that cuts off a receive event.

Specifications:

- Frequency tuning from a minimum frequency of 0.0000116415 Hz, in steps that size, up to the Nyquist limit (250 kHz). The DAC output, and ADC input should be low-pass filtered at a corner frequency no higher than 200 kHz.

- ADC input: source impedance. The source impedance should be < 10 kOhm. This comes from the TRACKTIM calculation in Sec. 45.7 of the SAM3X data sheet. With an 84 MHz clock frequency, the ADC clock is at 21 MHz. The tracking time is 15 clock cycles – so 714 ns. The formula there says: $Z_{source} < (t_{TRACK}-205)/0.054$ (times in ns). With 100 MHz clock speed, the ADC clock will run at 16 MHz, and the allowed impedance goes up to 13 kOhms.
- DAC output varies from 1/6 VCC to 5/6 VCC. If the output impedance of the DAC is described in the data sheet, it's well hidden. I'd present at least 1000 Ohms to the DAC output, and test it.
- Pulse program, total duration is presently 14000 32-bit words, but could be considerably longer if the downloadable waveform option is removed.
- Minimum event times. This is tricky. Transmit and Receive events are in 2us increments, and can be as short as 2us. Though having lots of these together in a row (eg in modulating the output phase very smoothly) may overload the arduino cpu. GPIO events on the other hand have ~24 ns granularity, with a minimum event time of about 1us. Again, stringing many of these together may overload the cpu. This is not well tested at this point.