Coherent Phase Shift Keying (PSK) Modulation Using Low-Power Micro-controllers for Underwater Acoustic Communications

Alexis Soulias, Yukang Xue, Y. Rosa Zheng, *Fellow, IEEE,*
Dept. of Electrical & Computer Engineering, Lehigh University, Bethlehem, PA 18015

OCEANS 2022, Hampton Roads | 978-1-6654-6809-1/22/$31.00 ©2022 IEEE | DOI: 10.1109/OCEANS47191.2022.9977244

***Abstract*—Phase Shift Keying is a single-carrier coherent modulation scheme that requires accurate phase and frequency control. We evaluate two approaches to accurately control the phase of the modulated signals for acoustic transmitters implemented by a low-power microcontroller (MCU) with a low system clock. One approach is to use one hardware timer for the carrier frequency and one software counter for the symbol rate. Another approach is to use two hardware timers: one for the carrier frequency and one for the symbol rate. We provide the detailed implementation on a Texas Instruments' MSP430F5529 MCU and compare the two approaches for BPSK and QPSK. Our testing results show that care has to be taken to ensure that the symbol phase control is accurate within 0.01% of the carrier frequency.**

***Index Terms*—PSK, underwater acoustic communication, microcontroller**

I. Introduction

Underwater sensors and data loggers are highly beneficial for the future Underwater Internet of Things, which have important applications in various areas, such as civil engineering studies in soil erosion, bridge scouring, and storm-water and sediment management [1]. Currently, most data loggers are placed underwater for data collection without transfer data in real time. They are often removed from the water to connect to a host computer via a cable, BlueTooth, or WiFi to transfer data to the internet [2], [3]. In all cases, these sensors or data loggers lack efficiency in real-time data collection. To achieve this real-time data transfer while working underwater, the sensors must be equipped with underwater wireless communication means, such as acoustic communication, magneto-inductive communication, or optical communications. However, most of the existing underwater wireless communication systems are quite large and expensive, making them difficult to integrate with small underwater sensors and data loggers.

In this project, we choose a high frequency acoustic transducer at 200 kHz to achieve short range underwater wireless communication. The TI's low-power MSP430F5529 microcontroller (MCU) is selected to interface with sensors and implement underwater acoustic communication with single-carrier coherent modulation (SCCM). In contrast to the implementation on sophisticated Field Programmable Gate Arrays (FPGA) [4] or Digital Signal Processors (DSP) [5], MCU implementation presents unique challenges due to its low system clocks and limited resources, especially when low-power consumption is a constraint. When setting the MCU's system clock as low as 24 MHz, there is a timing discrepancy in the high carrier frequency due to the intermediary time it takes for the hardware timer to change modes and the execution of its Interrupt Service Routine (ISR). This results in a phase drift or non-coherence of the carrier frequency in the modulated pass-band signal. This paper investigates two implementations to combat this problem and evaluates their performance; Additionally, it demonstrates each method's ability to show the same level of accuracy. Ultimately, it allows a user the flexibility in choosing a solution based on the resources available in a given microcontroller.

II. Modulation

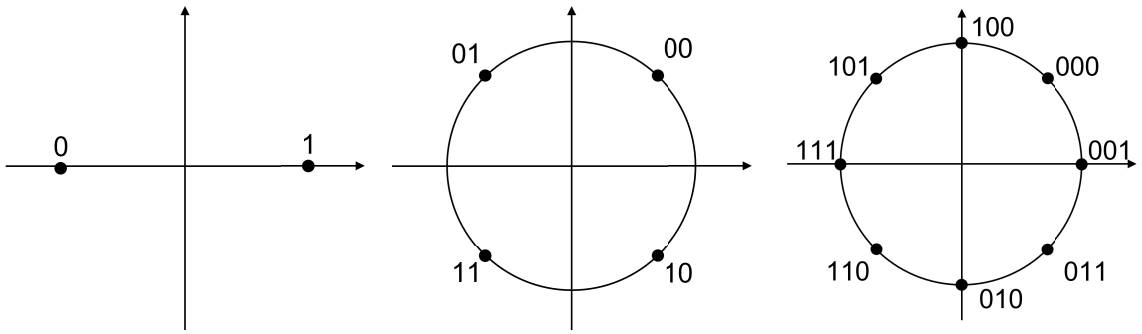
Modulation is the process of converting a digital bit stream to an analog signal at a high frequency, allowing the signal to transmit across physical mediums. Coherent phase shift keying usually maps the bits to complex symbols with Gray coding, as shown in Fig. 1. Binary phase shift key (BPSK) uses one input bit to select one of two phases which have a 180 degree difference. Similarly, the QPSK (or 8-PSK) modulation has four (or eight) different phases of the carrier waveform that are selected by the two (or three) input bits.

A. Implementing Binary Phase Shift Key

The TI MSP430 Microcontroller comes equipped with various pulse width modulation (PWM) output modes that can be used to implement the two different phase waveforms needed for BPSK [6]. The Timer_A module in the MCU generates the carrier frequency and then sets and controls the PWM outputs. As shown in fig. 2 the timer's clock signal is sourced by the SMCLK through a divider. The timer's counter is then compared to either the TAxCCR0 or TAxCCRn register, and when equal, the output mode is specified and enabled.

The given PWM outputs produce a square wave that is used for testing the carrier frequency and modulation. However, on the front-end transmitter side, there is a LC resonance circuit that produces a low-pass filter. This removes the high frequencies from the square wave, resulting in a standard sinusoidal wave for the carrier.

Table I shows all the different output modes in the MSP430 F5529 MCU. In this project, output modes three (set/reset) and seven (reset/set) were used to create two different waveforms, inherently representing the 0 and 1 bit.



(a) BPSK constellation (b) QPSK constellation (c) 8-PSK constellation

Fig. 1: Ideal constellations for PSK bit-to-symbol mapping with Gray coding

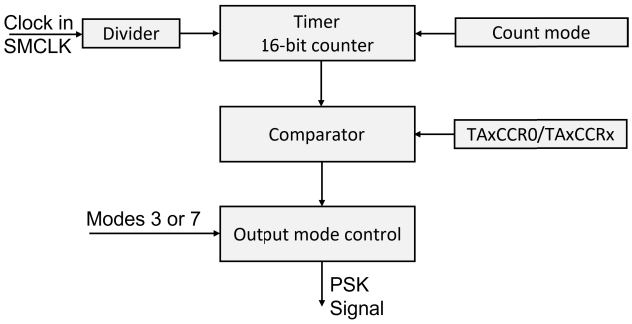


Fig. 2: Box Diagram depicting set up of the timer creating the carrier frequency and output modes.

TABLE I: Output modes of Timer_A module

Mode	Description
Mode 0	Output defined by OUT bit. Output updated immediately as OUT changes.
Mode 1	Output set when timer counts to TAxCCRn value. Remains set until timer is reset or until different output mode is selected.
Mode 2	Output set when timer counts to TAxCCRn value. Remains set until timer is reset or until different output mode is selected.
Mode 3	Output set when timer counts to TAxCCRn value. Resets when timer counts to TAxCCR0 value.
Mode 4	Output toggled when timer counts to TAxCCRn value. Output period is double timer period.
Mode 5	Output is reset when timer counts to TAxCCRn value. Remains reset until different output mode is selected.
Mode 6	Output is toggled when timer counts to TAxCCRn value. Output is set when timer counts to TAxCCR0 value.
Mode 7	Output is reset when timer counts to TAxCCRn value. Output is set when timer counts to TAxCCR0 value.

Fig. 3 shows the PWM output scheme, where register TAxCCR0 is set to create the carrier frequency f_c and TAxCCRn is set to control the duty cycle. Using the PWM 'UP' mode, the timer repeatedly counts from 0 up to the TAxCCR0 register value and generates an interrupt. The output toggles once when the timer equals the TAxCCRn register value and once again when the TAxCCR0 value is reached, thus forming a square waveform with f_c = (TAxCCR0 + 1)/f_{Tclk}, where f_{Tclk} denotes the frequency of the timer clock.

In our experiments, we use Timer_A0 and set the TA0CCR0 register to 119 and the corresponding TA0CCR1 register to 59, creating a duty cycle of 50% with a f_c of 200kHz. Depending on whether the transmitted bit is 0 or 1, we choose PWM output mode 3 or 7 to generate the square waveforms that are 180 degrees out of phase. With two square waves of opposite phases we produce the two symbols needed in BPSK. This process setups one hardware timer that is used in both proposed solutions.

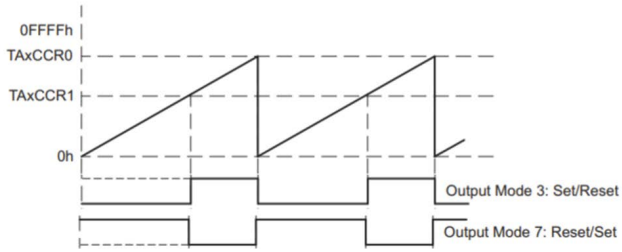


Fig. 3: PWM output scheme. The carrier frequency is determined by one cycle of counting up with timer register TAxCCR0.

B. Problems in the Initial Implementation

This paper will discuss the intricacies in implementing the two different methods to achieve accurate signal transmission. To precisely and consistently transmit bits, the cycles per symbol (CPS), needs to be well-defined. The CPS defines how many cycles of a particular phase must be counted in order to officially determine the bit as a 0 or 1: CPS = f_s/ f_c, where f_s denotes the symbol rate. In our project, utilizing a CPS of 10 means that to transmit a '0' bit, 10 cycles of the high waveform must be tracked and vice versa with the low waveform phase as the '1' bit.

In both solutions, two timers are needed: one to create the carrier frequency and one to count the CPS. When initially implemented, we observed that after each flip from one output mode to another, there were extra cycles being captured in each

bit. This was determined to be caused by the first timer that generates the 200kHz carrier frequency not stopping during the small time it takes for the second timer to disable an interrupt. The time it takes for the hardware processor to transition between counting the CPS for one bit and then starting the transmission over for the next bit in the signal leads to a timing issue as the extra cycles create a phase offset. When the phase offset becomes half a cycle, the waveforms invert, thus making the output for a 0 bit look like the waveform for a 1 bit and vice versa. When transmitting the same signal over and over, this timing issue and resulting phase difference make the output unreadable as the signal looks different each time.

III. Proposed Approaches

The general implementation utilizes the PWM output modes to create the standard square wave forms needed to represent a low wave and a high wave. These output modes are assigned respectively to the bits 0 and 1, and the mode changes correspondingly when the counting finishes in the second timer. This paper's experiments take this general implementation and look at different types of timers to use for calculating the CPS. Each solution has a hardware timer to create the carrier frequency, f_c, at 200kHz; however, the first solution uses a software timer to count the CPS while the second implementation uses another hardware timer instead. The focus will be on ensuring each method performs to the same standard.

The symbol rate or bandwidth of either approach can be written as:

$$\text{Bandwidth} = \frac{f_c}{n} \tag{1}$$

Fig. 4 shows the block diagram of the BPSK implementations with one hardware timer plus one software counter. In Fig. 5, we illustrate the block diagram of the QPSK with solution of two hardware timers.

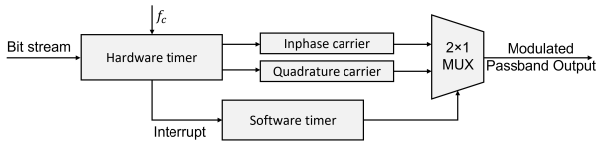


Fig. 4: Block diagram for one hardware timer plus one software counter in BPSK

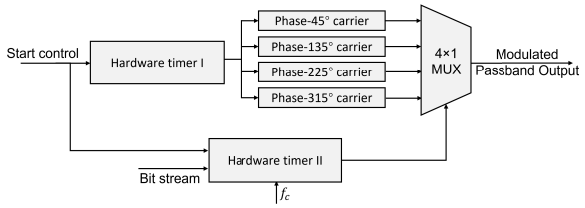


Fig. 5: Block diagram for two hardware timers in QPSK

The Software Timer Approach: The first approach uses the hardware Timer A0 setting TA0CCR0 register to 119 and the comparator register, TA0CCRn, to 59, which generates the 200kHz carrier frequency. A software timer is then implemented during the interrupt routine of this hardware timer to record the CPS with a precise control. The interrupt is enabled each time the counter in Timer A0 equals the TA0CCRn value.

In this interrupt routine, the software counter, n, gets incremented by one, allowing the counter to track and record the CPS of the current transmitted bit. When the counter n reaches 10, we end the transmission of the current bit, clear the counter, and start the transmission of the next bit. Notice that when the above counter n reaches 10, that means the software timer has counter to 59 (TA0CCRn) for the tenth time, but there are still 60 clock cycles before the tenth count to 119 (TA0CCR0). It is during the 60 clock cycles before this next symbol that the program completes the reconfiguration of the Timer A0 output mode.

Fig. 6 and Fig. 7 show the results of the software counter scheme, capturing images of the 200 kHz BPSK modulated output on an oscilloscope. The period between two adjacent bits is still the expected 5 microseconds, and the deviation of 0.56 may come from the accuracy of the oscilloscope, which is acceptable. The results shows the software counter scheme does not have any timing problem between two adjacent bits, proving to be an accurate method of implementation.

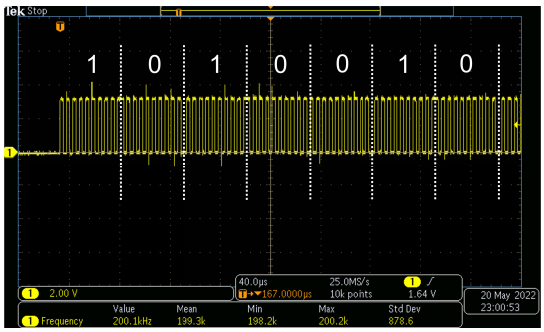
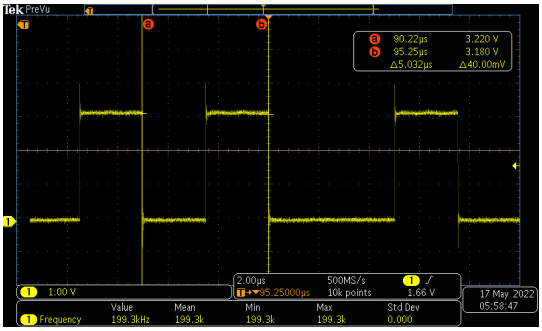
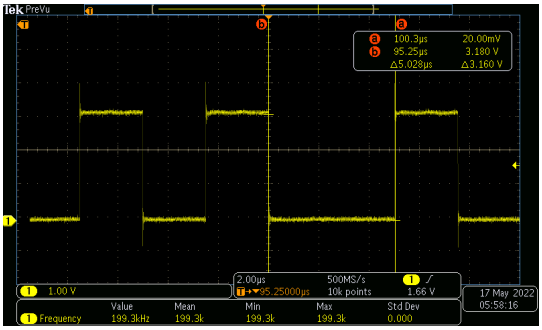


Fig. 6: Results of the software counter scheme.

The Two-hardware Timer Approach: This approach removes the need for a software timer and instead uses two hardware timers. The first hardware timer, A0, continues to generate the 200 kHz BPSK carrier signal; the second hardware counter has a count period that is an integer multiple of the first counter and is used to record CPS. In the first hardware counter, as shown in fig.2, the counter is sourced by the SMCLK with a divider of 1 to provide a 24 MHz clock source. Like in the previous solution, the count control mode is set to "up mode" and the register TA0CCR0 is set to 119 and TA0CCRn is set to 59 for the comparator. This creates the standard square wave at 200kHz and when enabled, will then configure the output modes 3 and 7 of the PWM to set high and low wave forms for the 0 and 1 bit. The second hardware timer, A1, is also driven by a 24MHz clock source and configured in "up mode." Its TA1CCR0 register is set to 1199 which is 10 times the counting period of A0; therefore, each complete cycle of timer A1 corresponds to 10 cycles



(a) A normal square wave period = 5 us with 0.64% deviation



(b) Period at adjacent bits = 5 us with 0.56% deviation

Fig. 7: Timing Results of the software counter scheme

of timer A0. This is how we are able to track the CPS with timer A2. Ideally, we would be able to start timers A0 and A1 at the same time, keeping them in sync to avoid timing discrepancies. If so, we know that in the process of counting 1199 from 0 in A1, A0 completes the process of counting from 0 to 119 10 times. After A0 has completed 9 counts from 0 to 119, we should be ready to switch to the next symbol to be transmitted. Essentially, while A0 is transmitting the tenth signal cycle, the program already knows that this is the last cycle representing the current symbol. Therefore, we set timer A1's register TA1CCRn to 1150, a number that lies between 1140 and 1199. When timer A1 counts to 1150, the program re configures the output mode of timer A0 based on the value of the next transmitted symbol. Since enough clock cycles are left between 1150 and 1199 to complete the reconfiguration of timer A0, we solve the timing problem that exists for switching between the two transmission symbols. Note that the value of register TA1CCRn for timer A1, if chosen too small, can cause distortion in the tenth cycle of the current transmitted symbol. 1150 is the appropriate choice based on measurements that leave enough clock cycles to complete the reconfiguration of the timer from the current symbol to the next symbol without causing severe distortion in the tenth cycle of the current symbol. Fig 8 shows the periods of each hardware timer, displaying how the two correlate.

In the process of actual programming, we found that the MSP430F5529 does not provide a function to enable two hardware timers at the same time. For this reason we had to start the counting of two registers one after the other. Experimental

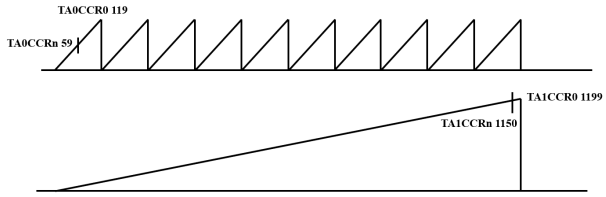


Fig. 8: Timer A0 is set with a period of 119 and a comparator of 59 to achieve a 50% duty cycle to create a 200kHz f_c. Timer A1 is set with a period of 1199 and a comparator of 1150 to count a full 10 cycles of the f_c while leaving enough time for the processor to transition between bits.

tests showed this sequentially enabled operation resulted in the timer that was enabled first counting a steady extra 17 clock cycles. After repeated tests, it was verified that these 17 clock cycles are always fixed, so we can still equivalently assume that timer A0 and timer A1 are synchronized but with a constant gap. The 17 clock cycle gap in timing can lead to inaccurate transmission of the first symbol. We observed that the initial bits were facing interference from this constant phase shift that took more than one bit but less than two, to fully correct. Fig.9 shows the interference in the initial bits.

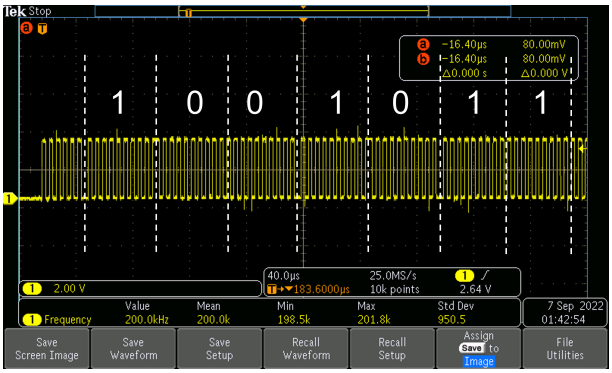


Fig. 9: Expecting the first nine bits to be 101001011 but only seeing 1001011 clearly as the first two bits get lost in the time difference between the two timers starting.

We measured the discrepancy and calculated that this phase interference was a constant 17 cycles each time. To combat that, we added an extra two bits to the front of each signal to act as parity bits. These two new starting bits buffer the initial phase difference and allow us to mark where the real signal starts, which enables a full, clean transmitted signal.

After adding the two extra bits at the start of each signal, we were able to properly transmit the signal to the correct CPS accuracy.

IV. Further Experiments: QPSK Implementation

After concluding that both solutions can be produced for BPSK modulation, we wanted to expand the testing to Quadrature Phase Shift Keying (QPSK). In QPSK there are four carrier phases with two bits being modulated at once to

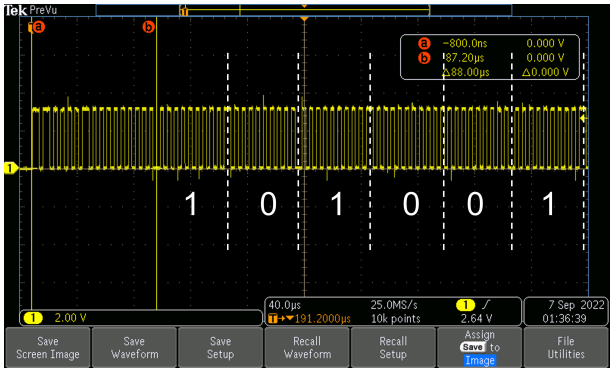
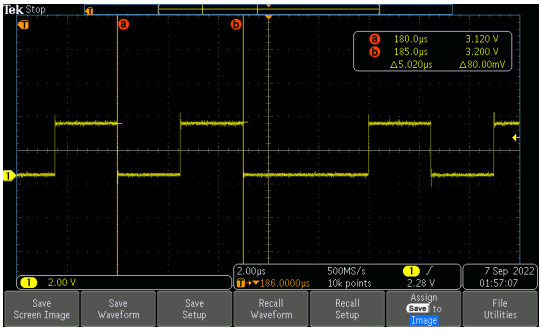
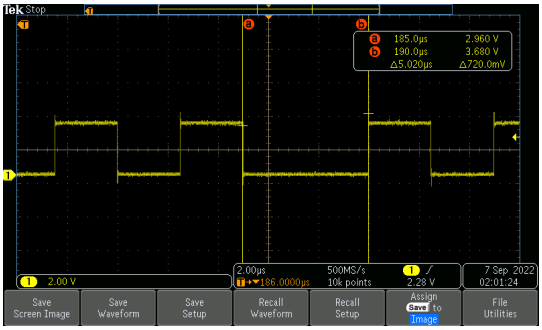


Fig. 10: Initial dummy bits were added to the beginning of each transmission and calculated to always find the distinct start of the true signal. The real signal starts after the yellow marker.



(a) A normal square wave period = 5 us with 0.64% deviation



(b) Period at adjacent bits = 5 us with 0.56% deviation

Fig. 11: Timing Results of the hardware counter scheme

determine the phase. This method of modulation allows for a signal to carry twice as much information as BPSK. In the following experiments, we aimed to prove that each method previously designed to fix the timing discrepancy in the BPSK modulation could be extended to use for more complicated modes of modulation too.

The details of implementing QPSK modulation have been provided in [7] however, the main idea is that the initial starting phase is controlled by two duty cycles and two PWM output modes. During the transmission of the signal, the PWM output mode 4 is strictly used to create the different phases. This general implementation was tailored to then use a software timer for the CPS counter in the first experiment and then a second hardware timer in the next. Both solutions integrated the same as previously in the BPSK with the software timer working as is and the second hardware timer needing the extra initial bits to handle the interference. The initial phase difference in the two hardware timer approach was calculated to be the same amount of constant time as before in the BPSK modulation. The following two figures, Fig.12 and Fig.13, display the results of each implementation properly transmitting the given signal.

In the software timer experiment we gave the signal 0231003. The yellow waveform is the actual signal, and the blue is a comparison waveform of a standard phase to allow easier detection of the four difference phases. For the two hardware timer experiment, we gave a different signal, 011133, and repeatedly sent it to ensure the timing discrepancy had been properly handled.

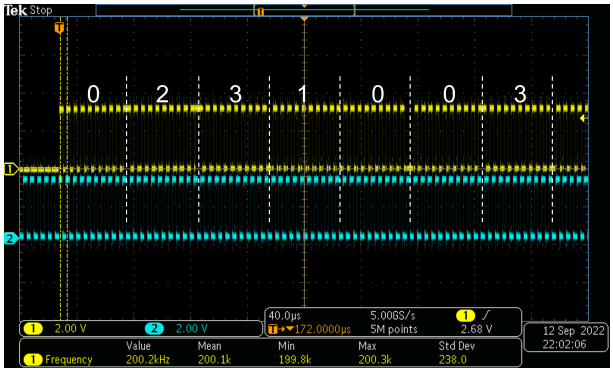


Fig. 12: QPSK implemented using one hardware timer and one software timer.

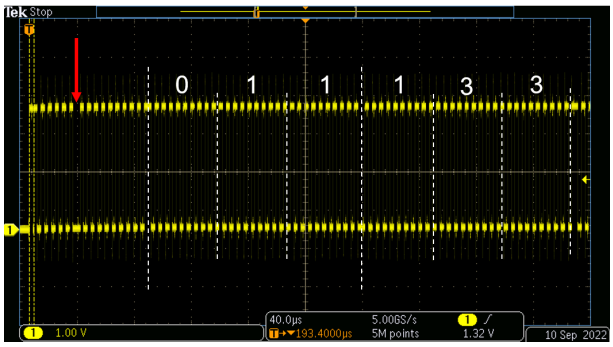


Fig. 13: QPSK implemented using two hardware timers.

V. Conclusion

Leveraging TI's low-power MSP430F559 microcontroller, we were able to implement BPSK modulation with a 200kHz carrier frequency under two different schemes. Hardware timer A0 was set in each solution to create f_c while a second timer, software or hardware was generated to count the CPS. Our results proved that both proposed solutions can be produced

within the necessary levels of accuracy to transmit an acoustic communication signal under BPSK modulation. We also proved that these implementations can be used to generate more complex modes of modulation, expanding both methods to work under QPSK, four phase, modulation. While each method does work, the software timer is an easier implementation as there is no extra timing discrepancy to worry about with having to start two timers synchronously. However, with the extra step of adding some parity bits before the real transmitted signal, a user could use the two hardware timer solution to the same affect, providing the intended flexibility in choosing available resources on a microcontroller.

ACKNOWLEDGMENT

This work is supported in part by the I-DISC undergraduate research program of Lehigh University and the NSF research grant IIP-1853258.

REFERENCES

[1] Y. R. Zheng, X. Zhu, and M. Tan, "Miniature underwater animal tags and smart sensors for civil engineering applications," in *Proceedings of the International Conference on Underwater Networks Systems*, ser. WUWNET'19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3366486.3366543

[2] Y. Kao, D. Frank, D. Foster, K. Huang, C. Kao, and P. H. Chou, "An in-situ motion measurement system for underwater sediments tracking," in *2014 IEEE International Conference on Internet of Things (iThings), and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom)*, Sep. 2014, pp. 360–367.

[3] D. Frank, D. Foster, P. Chou, Y.-M. Kao, J. Calantoni, and I.-M. Sou, "Direct measurements of sediment response to waves with 'smart sediment grains'," 10 2013, pp. 1–5.

[4] J. Li and Y. R. Zheng, "Hardware and software co-design of an underwater acoustic modem," in *MTS/IEEE Global Oceans 2021: San Diego – Porto*, 2021, pp. 1–6.

[5] Y. R. Zheng, Z. Yang, J. Hao, and P. Han, "Hardware implementation of underwater acoustic localization system for bridge scour monitoring," in *Oceans-San Diego, 2013*. IEEE, 2013, pp. 1–6.

[6] H. Qiu, "BPSK Modem Implementation With MSP432™ MCUs," 2016. [Online]. Available: https://e2e.ti.com/cfs-file/__key/communityserver-discussions-components-files/166/slaa681a.pdf

[7] Y. Xue and Y. R. Zheng, "ntation of high-order psk modulation for mimo acoustic modem using microcontrollers," in *MTS/IEEE Global Oceans 2021: San Diego – Porto*, 2021, pp. 1–6.