Multi-Channel SAE-J2716 (SENT) Decoder Using NHET

Anthony Seely *Texas Instruments MCU Automotive Systems*

The SAE-J2716 standard [1] is a product of the Data Communications Standards Committee under the Electrical Systems Group of the Motor Vehicle Council. This is the same committee that has produced standards for the widely adopted SAE-J1850, CAN, and LIN protocols. This application report provides an example of how the TMS570 NHET v2 may be used to implement a decoder for the SAE-J2716 (Feb 2008) protocol standard; the decoder presents the data and eliminates the need for CPU to do it. (Note that the standard has been updated again in Jan 2010 but not covered by this application note).

1 A Brief Introduction to SAE-J2716

SAE-J2716 is a lightweight protocol designed for communication between remote sensor units and electronic control units. Low sensor cost, increased sensor resolution and sample rates, and electro-magnetic compatibility are key concerns addressed by SAE-J2716.

The source encoding scheme used by SAE-J2716 is called Single Edge Nibble Transmission (SENT). The SENT encoder accepts nibbles (4 bits) of data at a time, and converts each nibble into a pulse on the data line which has a variable duration dependent on the data. The pulse duration is referenced from falling edge to falling edge on the data line, so it is not critical that the driving circuit have matched rise and fall times.

1.1 SAE-J2716 Physical Layer

The standard proposes a three-wire interface between sensor and ECU consisting of sensor power and ground plus a single data line. Signaling is accomplished by driving the voltage of the data line between (nominally) 5V and ground. Waveform transitions are shaped for electromagnetic compatibility purposes.

For the purpose of this application note, we assume that the ECU includes a physical layer interface outside the microcontroller which reduces the signaling to 3.3V LVCMOS levels and sharpens the waveform transitions.

1.2 SAE-J2716 Data Link Layer

Figure 1 contains an example of an SAE-J2716 message frame which consists of a calibration pulse, message header pulse, two data pulses, and a CRC4 pulse. The number of data pulses per message may vary depending on the transmitting device used, but is fixed for a given transmitting device.

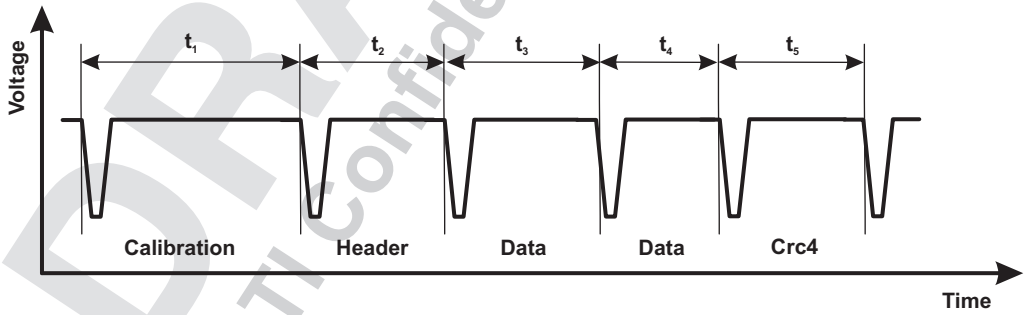


Figure 1. Example of an SAE-J2716 message frame

The duration of every pulse in the message frame is an integer multiple of a predetermined unit interval. The unit interval (UI) also varies but is fixed for a given transmitter. To enable low cost sensors, the transmitter is allowed a very wide tolerance (±20%) which necessitates starting each message with a training pulse. This is the function of the calibration pulse which has a duration of 56 UI (Equation 1)

$$t_{UI} = \frac{t_1}{56} \quad (1)$$

After the calibration pulse, the remaining pulses in the message represent nibble values. The pulse duration is proportional to the nibble value (N_i) plus an offset of 12 (Equation 2).

$$t_i = (12 + N_i) \cdot t_{UI} \quad (2)$$

The first nibble represents a message header. The main use of the message header is to encode a sub-channel that carries meta-data transmitted one bit per frame and decoded every sixteen frames.

After the message header, the data payload is transmitted as a series of data nibbles. The format of the payload is dependent on the definition of the transmitting device.

The last nibble is a CRC4 value that protects the header and data portions of the frame.

2 NHET Implementation of a SENT decoder

2.1 Timing Constraints

This section provides an overview of the timing constraints for the SENT decoder and how these constraints drive selection of the NHET clock prescaler values.

A general purpose SENT decoder needs to operate at up to the fastest data rate provided for in the SAE-J2716 standard (3µs unit interval or tick) and also take into account that the tolerance on the transmitter clock is ±25%. The worst case unit interval to design for is listed in Equation 3

$$t_{UI\_\min} = 2.25\ \mu s \quad (3)$$

A second requirement from the standard is for the decoder detect a shift greater than 1.5625% (1/64) between two successive calibration pulses. Assuming the transmitter operates at the maximum allowed clock rate, this means the NHET decoder must be designed to determine when two consecutive calibration pulses differ by more than 1.97us as calculated in Equation 4.

$$\frac{56}{64} t_{UI\_\min} = 1.97\ \mu s \quad (4)$$

Equation 3 and Equation 4 constrain the minimum NHET high resolution clock frequency when executing SENT decode. The NHET high resolution clock is generated by passing the NHET module clock, VLCK2, through a clock divider. The divider value is referred to as the High Resolution Pre-Scale (HRP).

The value of HRP should be minimized if the timer is to operate at maximum resolution. However the HRP setting is also a factor in determining the maximum number of clock cycles per high end timer loop available for program execution. This relationship is expressed by Equation 6.

$$Cycles / Loop = HRP \cdot LRP \quad (5)$$

Setting a budget of 256 cycles per loop for decoder execution, and understanding that the maximum value of LRP is 128, the minimum value of HRP is 2. With VCLK2 running at 100MHz in order to maximize the number of SENT channels that can be decoded on a single NHET timer, the resolution on the timer HRP is 20ns. This means that unit interval scale measurements of Equation 3 and Equation 4 can be made with accuracy of approximately 1%.

An estimate can be made of the maximum number of SENT channels to be decoded by a single NHET timer, assuming execution time not NHET memory is the limiting resource. In order for the NHET to decode a number of channels (N_{ch}) simultaneously, the decoder must be able to process all channels within the time it takes to transmit the shortest valid pulse. This constraint is expressed by Equation 6.

$$N_{ch} \cdot HRP \cdot LRP \cdot t_{cVCLK2} \le 12 \cdot t_{UI\_\min} \quad (6)$$

The left hand side of Equation 6 is an expression of the time required for the decoder loop to execute N_{ch} iterations and therefore process each channel once. The right hand side of the equation is an expression of the minimum valid input pulse duration. This corresponds to the transmission of a nibble with value 0, or pulse duration of 12 UI.

Using 256 cycles per loop and t_{cVCLK2} = 10ns, a single NHET can decode a maximum of 10 channels (Equation 7) at the maximum transmission rate of a 3µs - 25% unit interval. Again this assumes the limiting factor is NHET execution speed and not the amount of NHET program memory available.

$$N_{ch} \le \frac{12 \cdot 2.25\mu s}{256 \cdot 10ns}$$
$$N_{ch} \le 10.54 \quad (7)$$

2.2 Channel State Maintained by the Decoder

The example decoder maintains an array of state information for each of the n channels. The state variables maintained are:

- CHn_NEWP: New pulse measurement on channel 'n'. Units of NHET Hi-Res Clocks
- CHn_NCAL: Nominal value of calibration pulse - constant depending on sensor. Units of NHET Hi-Res Clocks
- CHn_LCAL: Last valid calibration pulse on channel 'n'. Units of NHET Hi-Res Clocks
- CHn_STATE: Protocol decoder state word. Includes working and initial values for data count, subchannel data count, and crc as well as state.
- CHn_MSGDAT: Working buffer (shift register) for message data reception.
- CHn_SCDAT: Working buffer (shift register) for subchannel data reception.
- CHn_MSGBUF: Output buffer for message data reception.
- CHn_SCBUF: Output buffer for subchannel data.

2.3 Two Level Loop Structure

The example NHET code for a SENT decoder is partitioned into a two level loop structure. The inner loop performs the high resolution input capture function on each of the N_{ch} input pins. This function is executed during every NHET loop resolution period. The outer loop processes the input captures of the inner loop. The outer loop is also executed once every loop resolution period, but it processes only one channel at a time. This means that it takes the NHET N_{ch}loop resolution periods to process each one of the input channels once. Figure 2 contains a flowchart of the example decoder NHET code.

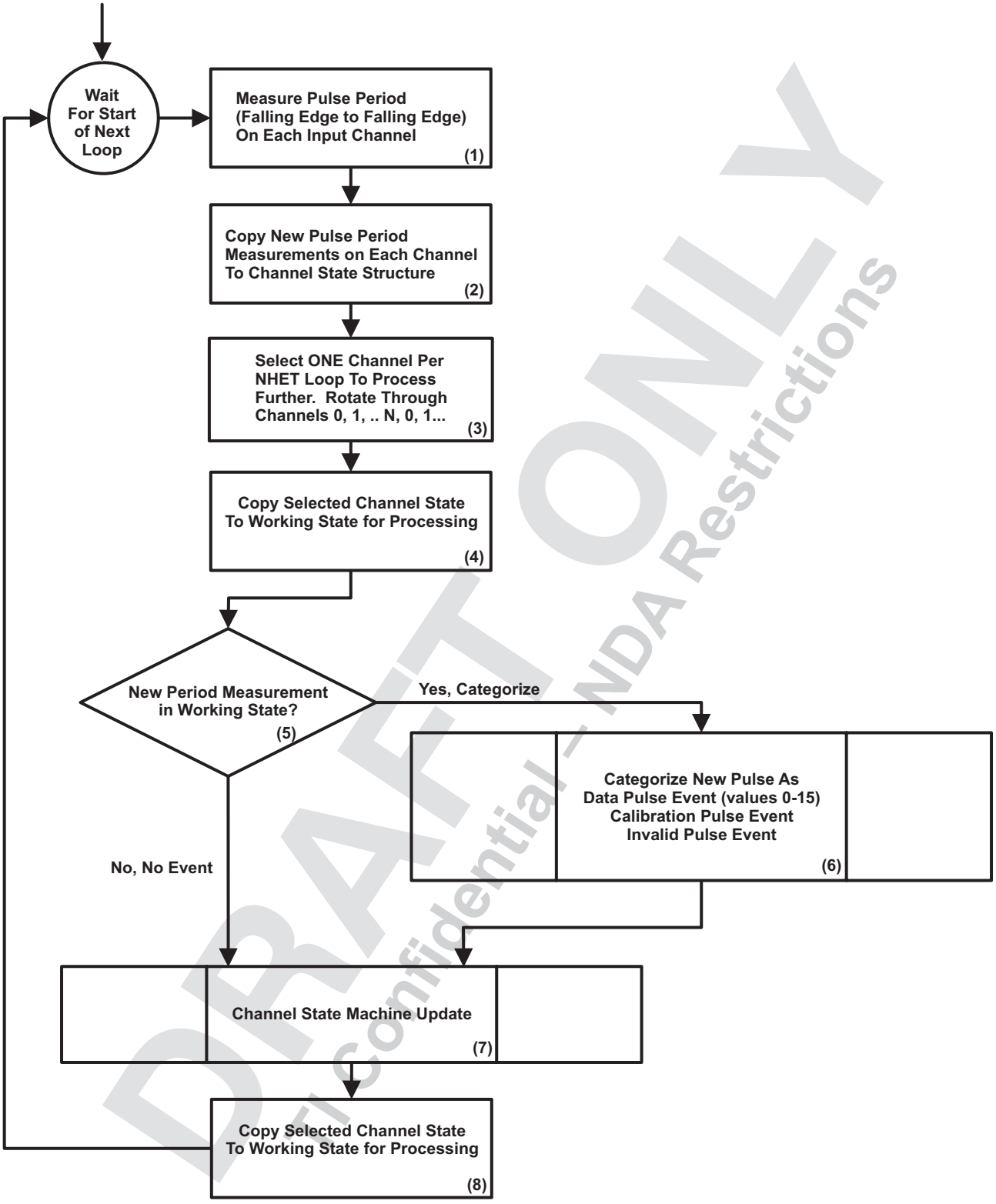
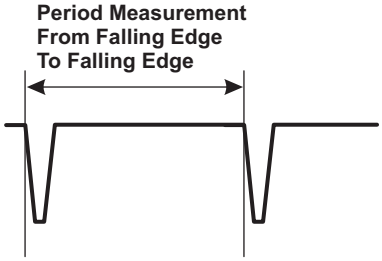


Figure 2. Flowchart of NHET SENT Decoder Example

2.4 Inner Loop: Pulse Period Measurement

Boxes (1) and (2) of Figure 2 represent the inner loop of input capture. The NHET instruction "PCNT" is used to implement an input capture function on each pin at high resolution (See Figure 3).



NHET Instruction To Measure Period at High Resolution On Input Pin NHET[n] (n={0,1,2,...}):

```
PCNT {hr_lr = HIGH, type=FALL2FALL, pin=n}
```

Figure 3. PCNT Instruction Used for Input Capture

Each PCNT instruction in the inner loop is followed by a pair of MOV32 instructions that execute conditionally each time a new input capture occurs in the preceding PCNT instruction. These two instructions copy the input capture result into the corresponding channel state data structure; where it is buffered for processing the next time that the corresponding channel is serviced in the outer loop.

The following code fragment performs the input capture function, and is repeated for each channel.

```
CH0_PCNT:
      PCNT {hr_lr=HIGH, type=FALL2FALL, pin=0}
      MOV32 {z_cond=ON, type=REMTOREG, reg=T, remote=CH0_PCNT}
      MOV32 {z_cond=ON, type=REGTOREM, reg=T, remote=CH0_NEWP}
```

2.5 Channel Dispatcher for Outer Loop

Boxes (3), (4), and (8) of Figure 2 represent the selection of one of the N_{ch} channels for processing in the outer loop.

Channel selection in Box (3) is performed with a CNT instruction followed by N_{ch}-1 ECMP instructions. The CNT instruction implements a modulo-counter that counts from zero to counter max value, then repeats again from zero. The ECMP instruction can be used to compare the counter value to each channel number and branch to the appropriate channel dispatch routine. The ECMP instruction has both a conditional address for compare match and next address (compare does not match) therefore only N_{ch}-1 ECMP instructions are required. An example dispatcher for a four channel decoder uses these NHET instructions:

```
L2DISPATCH:
              CNT {reg=A, max=3, data=0}
              ECMP {reg=A, cond_addr=CH3, pin=0, en_pin_action=OFF, data=3}
              ECMP {reg=A, cond_addr=CH2, pin=0, en_pin_action=OFF, data=2}
              ECMP {reg=A, cond_addr=CH1, pin=0, en_pin_action=OFF, data=1, next=CH0}
```

Once a channel is selected, the channel state is copied to a working scratchpad area that is used by the channel processing subroutine. The channel state is stored in the data field of the same instructions that are used to copy the channel state ot the working scratchpad area; saving NHET ram. An example of this NHET code for one channel is:

```
CH0:
PCH0_ADDR        .equ    00Ch
CH0_NEWP:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=L2WRK_NEWP,
        data=0, hr_data=0}
CH0_NCAL:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=L2WRK_NCAL,
```

```
        data=CONST_CH0NOMCALLR, hr_data=CONST_CH0NOMCALHR}
CH0_LCAL:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=L2WRK_LCAL,
        data=0, hr_data=0}
CH0_STATE:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=L2WRK_STATE,
        data=0, hr_data=0}
CH0_MSGDAT:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=L2WRK_MSGDAT,
        data=0, hr_data=0}
CH0_SCDAT:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=L2WRK_SCDAT,
        data=0, hr_data=0}
CH0_MSGBUF:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=L2WRK_MSGBUF,
        data=0, hr_data=0}
CH0_SCBUF:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=L2WRK_SCBUF,
        data=0, hr_data=0}
CH0_SURTN:
    ADD {src1=ZERO, src2=IMM, dest=T, rdest=REMP, remote=L2WRK_CLRNEWP,
        data=0, hr_data=PCH0_ADDR, next=L2_SETUP_RETURN}
```

Note that the last line of the above code fragment loads the tempory register 'T' with the address of the start of the channel data structure, then branches to "L2_SETUP_RETURN". The code at this address modifies the remote address of the instructions that copy the channel data structure back from the working / scratchpad area when processing of the outer loop is complete. This operation makes use of the remote program field option of the NHET v2 arithmetic instructions.

```
L2_SETUP_RETURN:
    ADD {src1=T, src2=IMM, dest=T, rdest=REMP, remote=L2WRK_NCAL, data=0, hr_data=1}
    ADD {src1=T, src2=IMM, dest=T, rdest=REMP, remote=L2WRK_LCAL, data=0, hr_data=1}
    ADD {src1=T, src2=IMM, dest=T, rdest=REMP, remote=L2WRK_STATE, data=0, hr_data=1}
    ADD {src1=T, src2=IMM, dest=T, rdest=REMP, remote=L2WRK_MSGDAT, data=0, hr_data=1}
    ADD {src1=T, src2=IMM, dest=T, rdest=REMP, remote=L2WRK_SCDAT, data=0  hr_data=1}
    ADD {src1=T, src2=IMM, dest=T, rdest=REMP, remote=L2WRK_MSGBUF, data=0, hr_data=1}
    ADD {src1=T, src2=IMM, dest=T, rdest=REMP, remote=L2WRK_SCBUF, data=0, hr_data=1}
```

Box (8) of Figure 2 is executed **after** all of the other processing in the outer loop is complete. This operation copies back the channel data structure from the scratchpad / working area to the appropriate channel data store. It is the code area that has remote addresses parameterized by the function L2_SETUP_RETURN in the above listing:

```
L2WRK_NEWP:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=CH0_NEWP,
        data=0, hr_data=0}
L2_RETURN:
L2WRK_CLRNEWP:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=CH0_NEWP,
        data=0, hr_data=0}
L2WRK_NCAL:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=CH0_NCAL,
        data=0, hr_data=0}
L2WRK_LCAL:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=CH0_LCAL,
        data=0, hr_data=0}
L2WRK_STATE:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=CH0_STATE,
        data=0, hr_data=0}
L2WRK_MSGDAT:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=CH0_MSGDAT,
        data=0, hr_data=0}
L2WRK_SCDAT:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=CH0_SCDAT,
        data=0, hr_data=0}
L2WRK_MSGBUF:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=CH0_MSGBUF,
```

```
        data=0, hr_data=0}
L2WRK_SCBUF:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=CH0_SCBUF,
        data=0, hr_data=0, next=PROGRAM_START}
```

As is the case with the channel state data structures, the working data is stored in the data field of the instructions that copy the scratchpad data back to the appropriate data structure. The data fields can be operated on in-place and restored simply by branching to L2_RETURN. Note that the CHn_NEWP state variable is not copied back but rather written back as a '0'. This is done to clear the CHn_NEWP so only one event is registered per incoming pulse.

2.6 Outer Loop: Pulse Categorization

Box (6) of Figure 2 is the pulse categorization step. This step is performed by performing a few simple checks and then executing a division. A more detailed flow chart of this step is illustrated by Figure 4. This step is bypassed when the L2WRK_NEWP is zero, so every time this routine is entered a new pulse event has been detected. Either the ratio of the new pulse period to the Last Calibration Pulse period or the Nominal Calibration Pulse Period is computed through division. This will depend on whether the L2WRK_LCAL state is non-zero; and the protocol state machine must manage this variable clearing it whenever an error is detected so that the nominal calibration pulse period is used whenever synchronization is lost. Once synchronization is achieved, incoming data and calibration pulses will be measured against the duration of the previous calibration pulse.

A division overflow would occur if the new pulse period (dividend) were greater than the divisor, so the division is performed using twice the calibration pulse period as a divisor. This accounts for incoming pulse periods that are slightly longer than the previous calibration pulse period as is allowed by the standard; but less than two times the previous calibration pulse period. The factor of 2 is divided back out in the last processing step (Box (6f),Figure 4) when the fractional result is normalized back to an integral number of unit intervals.

The division algorithm used is a sequential, restoring division algorithm described in [2].

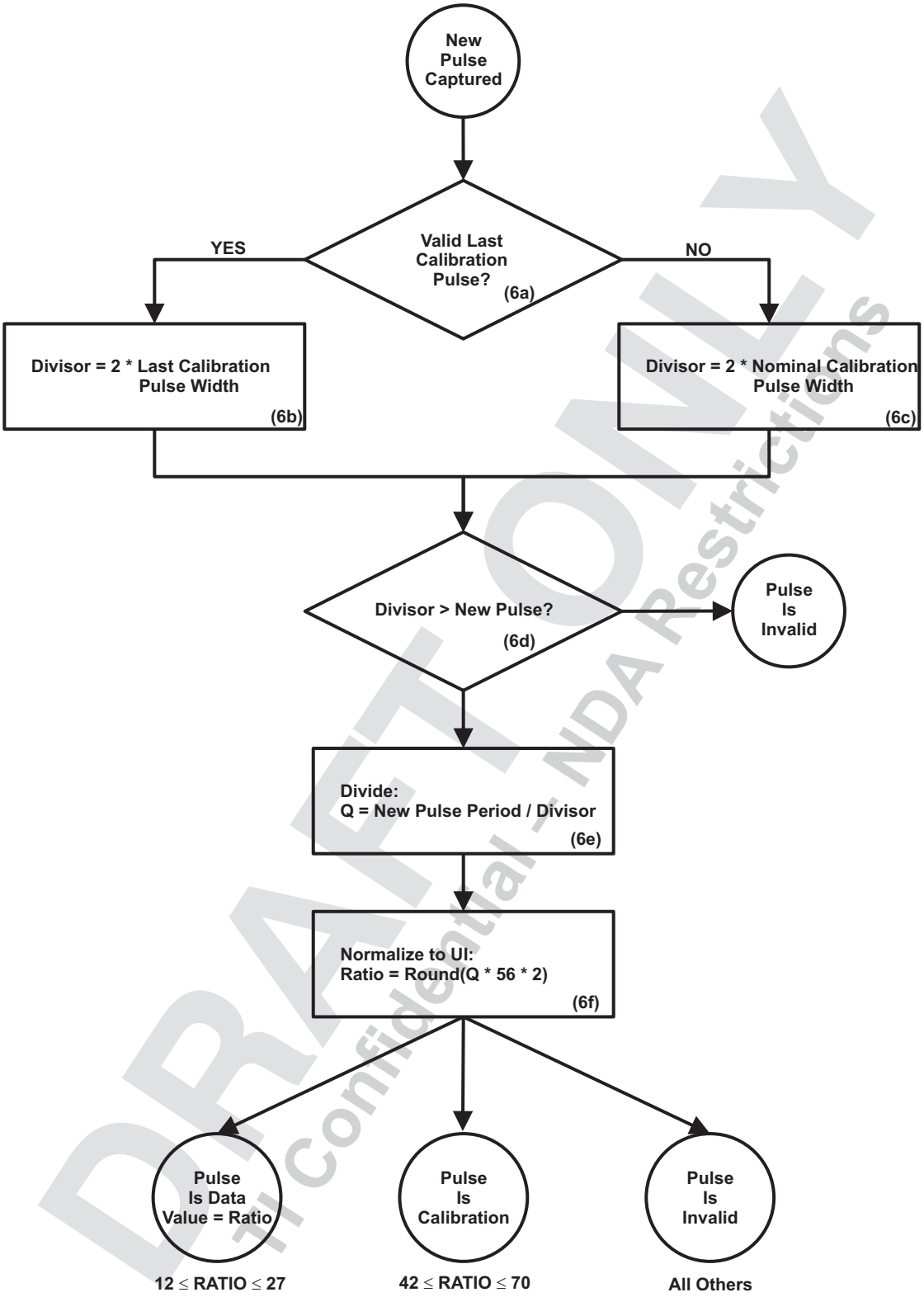


Figure 4. Pulse Categorization Flow Chart

The code for this section of the NHET algorithm follows. The division loop is iterated over 10 times based on empirical test results. The division loop body begins at L2_CHKD_A and ends at L2_CHKD_C. L2_CHKD_D is the beginning of the normalization step.

```
    ; Compute T = 56 * (NEWP / CAL)
    ; Step1:  T = NEWP / (2 * CAL).  Factor 2 to prevent division overflow
    ; Step2:  T = 56 * 2 * T
    ; Step3:  T = (2 * T) + 1
    ; Step5:  T = T / 2
    ; Result: T = ((56 * 2 * 2 * (NEWP / 2 * LCAL)) + 1) / 2 = 56 * NEWP / LCAL rounded to integer
L2_CHK:
    ; SETUP # OF ITERATIONS OF SRT DIVISION
    ; (10 Iterations, DJZ executes data+1 times, therefore data=9)
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=L2_CHKD_C, data=9}

    ; CLEAR QUOTIENT
    MOV32 {type=IMTOREG, reg=T, data=0, hr_data=0}

    ; DIVIDEND:  NEWP
    ; If New Pulse Width is '0' then no event was captured... skip.
    ADD   {src1=REM, src2=ZERO, dest=S, remote=L2WRK_NEWP, data=DUMMY}
    BR  {event=Z, cond_addr=L2_IS_NOEVENT}

    ; DIVISOR: (2 * CAL)  Where CAL is either LCAL or NCAL (if no valid 'last cal' exists)
    ADD {src1=REM, src2=ZERO, dest=R, remote=L2WRK_LCAL, data=DUMMY}
    BR  {event=NZ, cond_addr=L2_CHK_SETD}
    ADD {src1=REM, src2=ZERO, dest=R, remote=L2WRK_NCAL, data=DUMMY}

L2_CHK_SETD:
    ; Set Divisor in Division Loop
    ADD {src1=R, src2=R, dest=R, rdest=REM, remote=L2_CHKD_A, data=DUMMY}

    ; Confirm (2*CAL) > (NEWP).  If Not, Invalid
L2_CHK_OVFL:
    SUB {src1=R, src2=S, dest=IMM, data=DUMMY}
    BR  {event=N, cond_addr=L2_IS_INVALID}
    ADD {src1=S, src2=S, dest=S, data=DUMMY}

L2_CHKD_A:
    SUB {src1=S, src2=IMM, dest=R, data=DUMMY}
    BR  {event=N, cond_addr=L2_CHKD_B}

    ; SRT ALGO CONDITION 1: if 2r(i-1)-D is Positive or Zero
    ; then r(i) = 2r(i-1)-D and q(i) = 1
    ; Here, also multiply by '2' to pre-compute 2r(i) for next iteration
    ADD {src1=R, src2=R, dest=S, data=DUMMY}
    OR  {src1=IMM, src2=T, dest=T, smode=LSL, scount=1, data=0h, hr_data=1, next=L2_CHKD_C}

L2_CHKD_B:
    ; SRT ALGO CONDITION 2: if 2r(i-1)-D is Negative
    ; then r(i) = 2r(i-1), and q(i) = 0}
    ; Here, also multiply by '2' to pre-compute 2r(i) for next iteration
    ADD {src1=S, src2=S, dest=S, data=DUMMY}
    OR  {src1=IMM, src2=T, dest=T, smode=LSL, scount=1, data=0, hr_data=0}

L2_CHKD_C:
    DJZ {reg=NONE, cond_addr=L2_CHKD_D, next=L2_CHKD_A, data=9}

L2_CHKD_D:
    ADD {src1=ZERO, src2=T, dest=R, rdest=REM, remote=L2_CHKD_D, smode=LSL, scount=3,
        data=DUMMY}
    SUB {src1=R,    src2=T, dest=R, smode=LSL, scount=3, data=DUMMY}
    ADD {src1=IMM,  src2=R, dest=R, smode=LSR, scount=10, data=4h, hr_data=00h, next=dbga}
```

2.7 Outer Loop: Protocol State Machine

The outer loop protocol state machine is still to be implemented and tested. However a proposed state diagram is illustrated in Figure 5. In most cases when an error occurs the decoder should simply reset and attempt to resynchronize; so the state machine complexity is expected to be low.

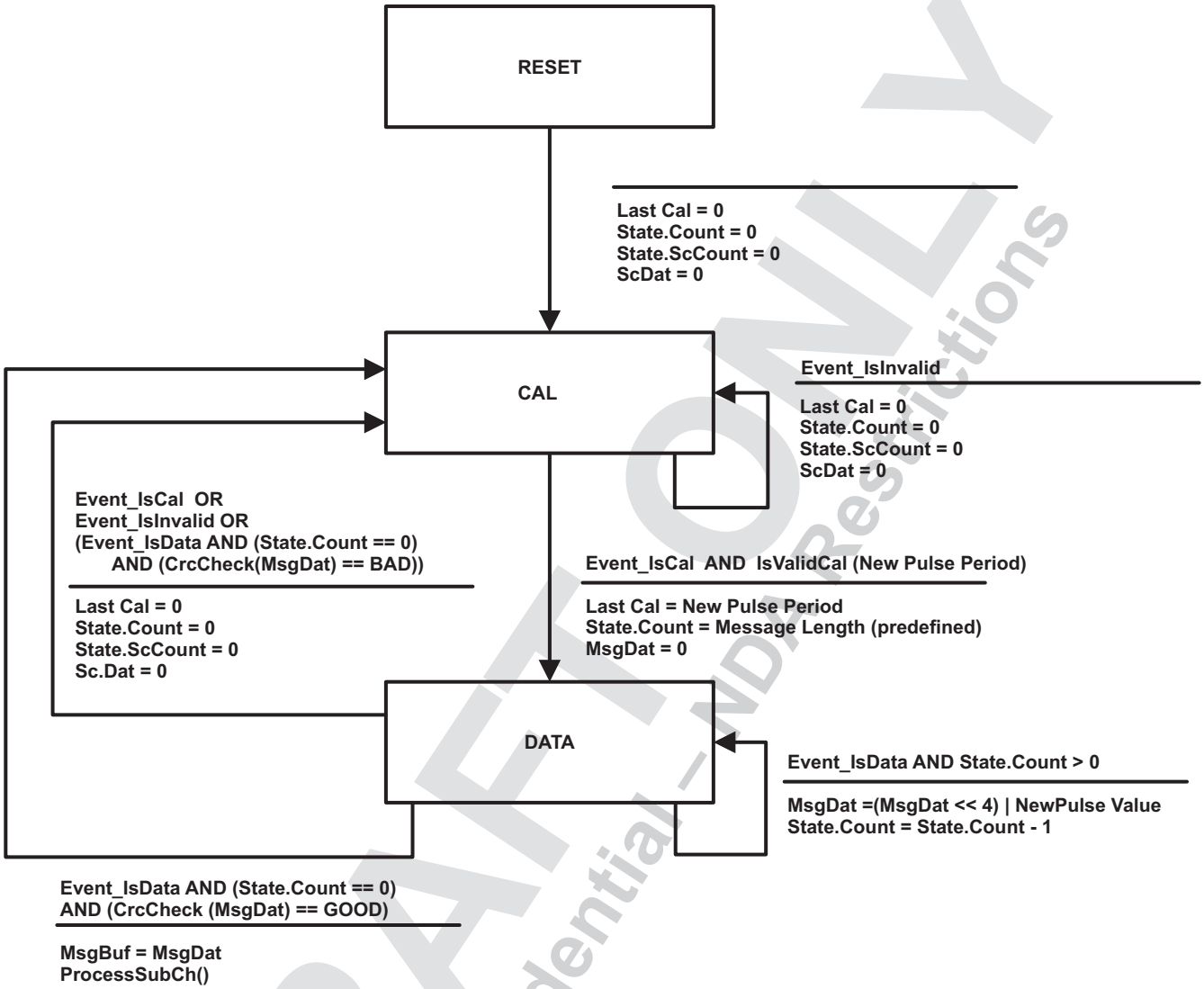


Figure 5. Proposed Outer Loop State Machine

2.7.1 CRC4 Check

The final nibble of every SENT message frame is a CRC4 that should be checked before declaring the message valid. The CRC4 polynomial from the Feburary 2008 standard is $x^4+x^3+x^2+1$ with a seed of 0x5. A table based MATLAB function is included in the standard as a reference.

We converted the table based, nibble at a time algorithm from the standard to a bit-serial form for implementation on NHET where RAM is very limited. The 'C' language subroutine for the serial approach is given in the following listing:

```
unsigned int CrcMthd2 (unsigned int *data, unsigned int numNibbles) {

    unsigned int i, bit_mask, temp_data = 0;
    unsigned int crc, poly;

    crc = CRCSeed;
```

```
    poly = 0xD;

    for (i = 0; i < numNibbles; i++) {
        temp_data = (temp_data << 4) | data[i];
    }

    bit_mask = 1 << ((numNibbles * 4) - 1);

    while (bit_mask > 0) {
        crc = crc << 1;
        if (bit_mask & temp_data) {
            crc = crc | 0x01;
        }
        if (crc & 0x10) {
            crc = crc ^ poly;
        }
        crc = crc & 0xF;
        bit_mask = bit_mask >> 1;
    }
    return crc;
}
```

The above code was tested on a PC against the table based form for all 16-bit inputs in the range 0x0000 to 0xFFFF. The equivalent NHET code is listed below (not yet tested - but included for cycle and memory usage estimation). This code assumes register R contains the working CRC value, and S contains the received data nibble. Register T is used for the bit mask. The working CRC should be stored in a nibble of the CHn_STATE state variable and initialized to CRC4_SEED upon detection of a new calibration pulse.

```
CRC4_SEED .equ 05h
CRC4_POLY    .equ 0Dh

    MOV32 {type=IMTOREG, reg=T, data=0, hr_data=8h}
CRC4:
    ADD {src1=R, src2=R, dest=R, data=DUMMY}
    AND {src1=S, src2=T, dest=NONE}
    BR {event=Z, cond_addr=CRC4_a}
    OR {src1=R, src2=IMM, dest=R, data=0, hr_data=1h}
CRC4_a:
    AND {src1=R, src2=IMM, dest=NONE, data=0, hr_data=10h}
    BR {event=Z, cond_addr=CRC4_b}
    XOR {src1=R, src2=IMM, dest=R, data=0, hr_data=CRC4_POLY}
CRC4_b:
    AND {src1=R, src2=IMM, dest=R, data=0, hr_data=0Fh}
    ADD {src1=T, src2=ZERO, dest=T, smode=LSR, scount=1, data=DUMMY}
    BR {event=NZ, cond_addr=CRC4}
```

3 Testing

The decoder has been implemented except for Step 7 of Figure 2; i.e. the decoder currently functions as a multichannel pulse categorization algorithm.

Testing of this functionality was performed with:

- FPGA board containing an TMS570 CPU, System, and NHET peripheral
- Tektronix AWG2021 Arbitrary Waveform Generator
- Tektronix TDS3014B Digital Oscilloscope

The FPGA board operates with the NHET timer VCLK2 at 15MHz; but input stimulus has been scaled accordingly to simulate VCLK2 at 100MHz. A Python scripts was written to generate simulated SENT waveform and program this into the arbitrary waveform generator as stimulus to the FPGA board. The scripts also allow the unit interval for each of the calibration, data, and CRC pulses to be varied from the nominal value.

Testing was done on the pulse categorization routine to verify the transition points between decode of one data code word to the next.This is done by keeping the calibration pulse period at a nominal value (scale factor of 1.0) while varying the scale factor of the data pulse and monitoring the NHET RAM location that contains the output code word (dbgr in the code listing of Section 6). The pulse width recorded is within the transition band and at this pulse width the decoder will sometimes decode the lower value symbol and sometimes the higher value symbol.

Results of the test on input channel 0 are listed in Table 1. Transition points are in the expected range.

Table 1. Pulse Categorization Testing Results on FPGA - 1120µs Calibration Pulse

Data Pulse Width	Receiver Transition (Nibble Received)	Expected Transition Point (µs)	Measured Transition Point (µs)
11.5 UI	Invalid -> 0h	230	231.8
12.5 UI	0h -> 1h	250	250.8
13.5 UI	1h -> 2h	270	271.2
14.5 UI	2h -> 3h	290	291.0
15.5 UI	3h -> 4h	310	310.5
16.5 UI	4h -> 5h	330	330.5
17.5 UI	5h -> 6h	350	350.0
18.5 UI	6h -> 7h	370	371.3
19.5 UI	7h -> 8h	390	391.8
20.5 UI	8h -> 9h	410	411.3
21.5 UI	9h -> Ah	430	431.0
22.5 UI	Ah -> Bh	450	450.8
23.5 UI	Bh -> Ch	470	470.0
24.5 UI	Ch -> Dh	490	489.2
25.5 UI	Dh -> Eh	510	511.2
26.5 UI	Eh -> Fh	530	531.5
27.5 UI	Fh -> Invalid	550	551.5

4 Code Size and Execution Time

The NHET RAM used by the currently implemented portion of the decoder is 88 words, not including the event counters and debug code that will be removed in the final implementation. The CRC4 algorithm will add 11 additional words, bringing the RAM usage for 4 channels to 99 words. This leaves just over 60 words available for implementing the proposed decoder state machine of Section 2.7 on a TMS570 device with 160 words of NHET RAM.

The incremental memory cost per additional channel is estimated to be 13 words: 3 words for event detection, 9 words for state, and 1 word for dispatch.

The cycle count for the currently implemented portion of the decoder is 124 cycles (based on manual instruction counting for the longest path). Adding a CRC4 check is another 44 cycles, bringing the total to 168 cycles per NHET loop. With a budget of 256 cycles (See Section 2.1) just under 100 cycles are left to implement the state machine proposed in Section 2.7.

Also, if necessary, subchannel processing can be performed as an additional state after the CRC word is received; to spread its cycle count over another loop resolution period. This can be done because after the CRC word is received, the only valid input pulse can be a calibration pulse, which is more than four times longer than the code-word 0 pulse from which the 256 word loop resolution budget is derived.

5 References

1. SAE International Surface Vehicle Information Report, "SENT -- Single Edge Nibble Transmission for Automotive Applications,"J2716, Rev. Feb. 2008.
2. Koren, Israel. "Computer Arithmetic Algorithms, Second Edition." A K Peters. 2002. Books24x7.

http://common.books24x7.com/book/id_15574/book.asp

6 Complete Code Listing

```
CONST_CH0NOMCALLR    .equ    20h
CONST_CH0NOMCALHR    .equ    68h
CONST_CH1NOMCALLR    .equ    20h
CONST_CH1NOMCALHR    .equ    69h
CONST_CH2NOMCALLR    .equ    20h
CONST_CH2NOMCALHR    .equ    6ah
CONST_CH3NOMCALLR    .equ    20h
CONST_CH3NOMCALHR    .equ    6bh

CONST_EVENT_NONE     .equ 16
CONST_EVENT_VALIDCAL .equ 17
CONST_EVENT_ERRCAL   .equ 18
CONST_EVENT_INVALID  .equ 19

PROGRAM_START:

CH0_PCNT:
    PCNT {hr_lr=HIGH, type=FALL2FALL, pin=0}
    MOV32 {z_cond=ON, type=REMTOREG, reg=T, remote=CH0_PCNT}
    MOV32 {z_cond=ON, type=REGTOREM, reg=T, remote=CH0_NEWP}

CH1_PCNT:
    PCNT {hr_lr=HIGH, type=FALL2FALL, pin=1}
    MOV32 {z_cond=ON, type=REMTOREG, reg=T, remote=CH1_PCNT}
    MOV32 {z_cond=ON, type=REGTOREM, reg=T, remote=CH1_NEWP}

CH2_PCNT:
    PCNT {hr_lr=HIGH, type=FALL2FALL, pin=2}
    MOV32 {z_cond=ON, type=REMTOREG, reg=T, remote=CH2_PCNT}
    MOV32 {z_cond=ON, type=REGTOREM, reg=T, remote=CH2_NEWP}

CH3_PCNT:
    PCNT {hr_lr=HIGH, type=FALL2FALL, pin=3}
    MOV32 {z_cond=ON, type=REMTOREG, reg=T, remote=CH3_PCNT}
    MOV32 {z_cond=ON, type=REGTOREM, reg=T, remote=CH3_NEWP, next=L2DISPATCH}

CH0:
PCH0_ADDR        .equ    00Ch
CH0_NEWP:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=L2WRK_NEWP,
        data=0, hr_data=0}
CH0_NCAL:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=L2WRK_NCAL,
        data=CONST_CH0NOMCALLR, hr_data=CONST_CH0NOMCALHR}
CH0_LCAL:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=L2WRK_LCAL,
        data=0, hr_data=0}
CH0_STATE:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=L2WRK_STATE,
        data=0, hr_data=0}
CH0_MSGDAT:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=L2WRK_MSGDAT,
        data=0, hr_data=0}
CH0_SCDAT:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=L2WRK_SCDAT,
        data=0, hr_data=0}
CH0_MSGBUF:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=L2WRK_MSGBUF,
        data=0, hr_data=0}
CH0_SCBUF:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=L2WRK_SCBUF,
```

```
        data=0, hr_data=0}
CH0_SURTN:
    ADD {src1=ZERO, src2=IMM, dest=T, rdest=REMP, remote=L2WRK_CLRNEWP,
        data=0, hr_data=PCH0_ADDR, next=L2_SETUP_RETURN}

CH1:
PCH1_ADDR        .equ     015h
CH1_NEWP:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=L2WRK_NEWP,
        data=0, hr_data=0}
CH1_NCAL:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=L2WRK_NCAL,
        data=CONST_CH1NOMCALLR, hr_data=CONST_CH1NOMCALHR}
CH1_LCAL:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=L2WRK_LCAL,
        data=0, hr_data=0}
CH1_STATE:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=L2WRK_STATE,
        data=0, hr_data=0}
CH1_MSGDAT:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=L2WRK_MSGDAT,
        data=0, hr_data=0}
CH1_SCDAT:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=L2WRK_SCDAT,
        data=0, hr_data=0}
CH1_MSGBUF:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=L2WRK_MSGBUF,
        data=0, hr_data=0}
CH1_SCBUF:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=L2WRK_SCBUF,
        data=0, hr_data=0}
CH1_SURTN:
    ADD {src1=ZERO, src2=IMM, dest=T, rdest=REMP, remote=L2WRK_CLRNEWP,
        data=0, hr_data=PCH1_ADDR, next=L2_SETUP_RETURN}

CH2:
PCH2_ADDR        .equ     01Eh
CH2_NEWP:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=L2WRK_NEWP,
        data=0, hr_data=0}
CH2_NCAL:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=L2WRK_NCAL,
        data=CONST_CH2NOMCALLR, hr_data=CONST_CH2NOMCALHR}
CH2_LCAL:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=L2WRK_LCAL,
        data=0, hr_data=0}
CH2_STATE:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=L2WRK_STATE,
        data=0, hr_data=0}
CH2_MSGDAT:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=L2WRK_MSGDAT,
        data=0, hr_data=0}
CH2_SCDAT:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=L2WRK_SCDAT,
        data=0, hr_data=0}
CH2_MSGBUF:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=L2WRK_MSGBUF,
        data=0, hr_data=0}
CH2_SCBUF:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=L2WRK_SCBUF,
        data=0, hr_data=0}
CH2_SURTN:
    ADD {src1=ZERO, src2=IMM, dest=T, rdest=REMP, remote=L2WRK_CLRNEWP,
        data=0, hr_data=PCH2_ADDR, next=L2_SETUP_RETURN}

CH3:
```

```
PCH3_ADDR        .equ     027h
CH3_NEWP:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=L2WRK_NEWP,
        data=0, hr_data=0}
CH3_NCAL:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=L2WRK_NCAL,
        data=CONST_CH3NOMCALLR, hr_data=CONST_CH3NOMCALHR}
CH3_LCAL:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=L2WRK_LCAL,
        data=0, hr_data=0}
CH3_STATE:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=L2WRK_STATE,
        data=0, hr_data=0}
CH3_MSGDAT:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=L2WRK_MSGDAT,
        data=0, hr_data=0}
CH3_SCDAT:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=L2WRK_SCDAT,
        data=0, hr_data=0}
CH3_MSGBUF:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=L2WRK_MSGBUF,
        data=0, hr_data=0}
CH3_SCBUF:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=L2WRK_SCBUF,
        data=0, hr_data=0}
CH3_SURTN:
    ADD {src1=ZERO, src2=IMM, dest=T, rdest=REMP, remote=L2WRK_CLRNEWP,
        data=0, hr_data=PCH3_ADDR, next=L2_SETUP_RETURN}


L2WRK_NEWP:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=CH0_NEWP,
        data=0, hr_data=0}
L2_RETURN:
L2WRK_CLRNEWP:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=CH0_NEWP,
        data=0, hr_data=0}
L2WRK_NCAL:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=CH0_NCAL,
        data=0, hr_data=0}
L2WRK_LCAL:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=CH0_LCAL,
        data=0, hr_data=0}
L2WRK_STATE:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=CH0_STATE,
        data=0, hr_data=0}
L2WRK_MSGDAT:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=CH0_MSGDAT,
        data=0, hr_data=0}
L2WRK_SCDAT:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=CH0_SCDAT,
        data=0, hr_data=0}
L2WRK_MSGBUF:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=CH0_MSGBUF,
        data=0, hr_data=0}
L2WRK_SCBUF:
    MOV32 {type=IMTOREG&REM, reg=NONE, remote=CH0_SCBUF,
        data=0, hr_data=0, next=PROGRAM_START}

L2DISPATCH:
    CNT {reg=A, max=3, data=0}
    ECMP {reg=A, cond_addr=CH3, pin=0, en_pin_action=OFF, data=3}
    ECMP {reg=A, cond_addr=CH2, pin=0, en_pin_action=OFF, data=2}
    ECMP {reg=A, cond_addr=CH1, pin=0, en_pin_action=OFF, data=1, next=CH0}

L2_SETUP_RETURN:
```

```
        ADD {src1=T, src2=IMM, dest=T, rdest=REMP, remote=L2WRK_NCAL, data=0, hr_data=1}
        ADD {src1=T, src2=IMM, dest=T, rdest=REMP, remote=L2WRK_LCAL, data=0, hr_data=1}
        ADD {src1=T, src2=IMM, dest=T, rdest=REMP, remote=L2WRK_STATE, data=0, hr_data=1}
        ADD {src1=T, src2=IMM, dest=T, rdest=REMP, remote=L2WRK_MSGDAT, data=0, hr_data=1}
        ADD {src1=T, src2=IMM, dest=T, rdest=REMP, remote=L2WRK_SCDAT, data=0  hr_data=1}
        ADD {src1=T, src2=IMM, dest=T, rdest=REMP, remote=L2WRK_MSGBUF, data=0, hr_data=1}
        ADD {src1=T, src2=IMM, dest=T, rdest=REMP, remote=L2WRK_SCBUF, data=0, hr_data=1}


        ; Compute T = 56 * (NEWP / CAL)
        ; Step1:  T = NEWP / (2 * CAL).  Factor 2 to prevent division overflow
        ; Step2:  T = 56 * 2 * T
        ; Step3:  T = (2 * T) + 1
        ; Step5:  T = T / 2
        ; Result: T = ((56 * 2 * 2 * (NEWP / 2 * LCAL)) + 1) / 2 = 56 * NEWP / LCAL rounded to integer
    L2_CHK:
        ; SETUP # OF ITERATIONS OF SRT DIVISION
        ; (10 Iterations, DJZ executes data+1 times, therefore data=9)
        MOV32 {type=IMTOREG&REM, reg=NONE, remote=L2_CHKD_C, data=9}

        ; CLEAR QUOTIENT
        MOV32 {type=IMTOREG, reg=T, data=0, hr_data=0}

        ; DIVIDEND:  NEWP
        ; If New Pulse Width is '0' then no event was captured... skip.
        ADD   {src1=REM, src2=ZERO, dest=S, remote=L2WRK_NEWP, data=DUMMY}
        BR  {event=Z, cond_addr=L2_IS_NOEVENT}

        ; DIVISOR: (2 * CAL)  Where CAL is either LCAL or NCAL (if no valid 'last cal' exists)
        ADD {src1=REM, src2=ZERO, dest=R, remote=L2WRK_LCAL, data=DUMMY}
        BR  {event=NZ, cond_addr=L2_CHK_SETD}
        ADD {src1=REM, src2=ZERO, dest=R, remote=L2WRK_NCAL, data=DUMMY}

    L2_CHK_SETD:
        ; Set Divisor in Division Loop
        ADD {src1=R, src2=R, dest=R, rdest=REM, remote=L2_CHKD_A, data=DUMMY}

        ; Confirm (2*CAL) > (NEWP).  If Not, Invalid
    L2_CHK_OVFL:
        SUB {src1=R, src2=S, dest=IMM, data=DUMMY}
        BR  {event=N, cond_addr=L2_IS_INVALID}
        ADD {src1=S, src2=S, dest=S, data=DUMMY}

    L2_CHKD_A:
        SUB {src1=S, src2=IMM, dest=R, data=DUMMY}
        BR  {event=N, cond_addr=L2_CHKD_B}

        ; SRT ALGO CONDITION 1: if 2r(i-1)-D is Positive or Zero
        ; then r(i) = 2r(i-1)-D and q(i) = 1
        ; Here, also multiply by '2' to pre-compute 2r(i) for next iteration
        ADD {src1=R, src2=R, dest=S, data=DUMMY}
        OR  {src1=IMM, src2=T, dest=T, smode=LSL, scount=1, data=0h, hr_data=1, next=L2_CHKD_C}

    L2_CHKD_B:
        ; SRT ALGO CONDITION 2: if 2r(i-1)-D is Negative
        ; then r(i) = 2r(i-1), and q(i) = 0}
        ; Here, also multiply by '2' to pre-compute 2r(i) for next iteration
        ADD {src1=S, src2=S, dest=S, data=DUMMY}
        OR  {src1=IMM, src2=T, dest=T, smode=LSL, scount=1, data=0, hr_data=0}

    L2_CHKD_C:
        DJZ {reg=NONE, cond_addr=L2_CHKD_D, next=L2_CHKD_A, data=9}

    L2_CHKD_D:
        ADD {src1=ZERO, src2=T, dest=R, rdest=REM, remote=L2_CHKD_D, smode=LSL, scount=3,
            data=DUMMY}
```

```
    SUB {src1=R,    src2=T, dest=R, smode=LSL, scount=3, data=DUMMY}
    ADD {src1=IMM,  src2=R, dest=R, smode=LSR, scount=10, data=4h, hr_data=00h, next=dbga}


L2_IS_DATA:
    CNT {reg=A, max=1FFFFFFh, data=0, next=L2_RETURN}
L2_IS_NOEVENT:
    CNT {reg=A, max=1FFFFFFh, data=0, next=L2_RETURN}
L2_IS_VALIDCAL:
    CNT {reg=A, max=1FFFFFFh, data=0, next=L2_RETURN}
L2_IS_ERRCAL:
    CNT {reg=A, max=1FFFFFFh, data=0, next=L2_RETURN}
L2_IS_INVALID:
    CNT {reg=A, max=1FFFFFFh, data=0, next=L2_RETURN}

               ; Debug Only
dbga:           MOV32 {type=REGTOREM, reg=A, remote = dbga}
dbgb:           MOV32 {type=REGTOREM, reg=B, remote = dbgb}
dbgr:           MOV32 {type=REGTOREM, reg=R, remote = dbgr}
dbgs:           MOV32 {type=REGTOREM, reg=S, remote = dbgs}
dbgt:           MOV32 {type=REGTOREM, reg=T, remote = dbgt, next=L2_IS_DATA}
```

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers’ products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers’ products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers’ products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers’ products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265