DS5101 Board Reference

Document Version 1.2.1

dSPACE GmbH
Rathenaustraße 26
33102 Paderborn
Germany

dSPACE GmbH
Rathenaustraße 26
D-33100 Paderborn
Germany

Table of Contents

Table of Figures

Table of Tables

1 Introduction

The DS5101 Digital Waveform Output (DWO) board is a member of the modular dSPACE system specifically designed for high speed signal generation. It can be used for fast and complex waveform generation on up to 16 channels. It’s key features are:

- Modular system with motherboard and up to eight modules each containing 2 channels (from version DS5101-04: Eight modules integrated within baseboard)
- 25 ns resolution
- Shortest pulse length is 250ns
- Delays from 250ns up to 26.8 seconds
- Up to 512 different delay values
- Delays can be interrupted for shorter delay values (programmable)
- Up to 128 program steps
- Conditional branches on flags, loop counter or other channel I/O
- Output set, reset, toggle
- Trigger on rising, falling or both edges
- All channels can trigger each other
- Software trigger from PHS-bus
- PHS-bus interface

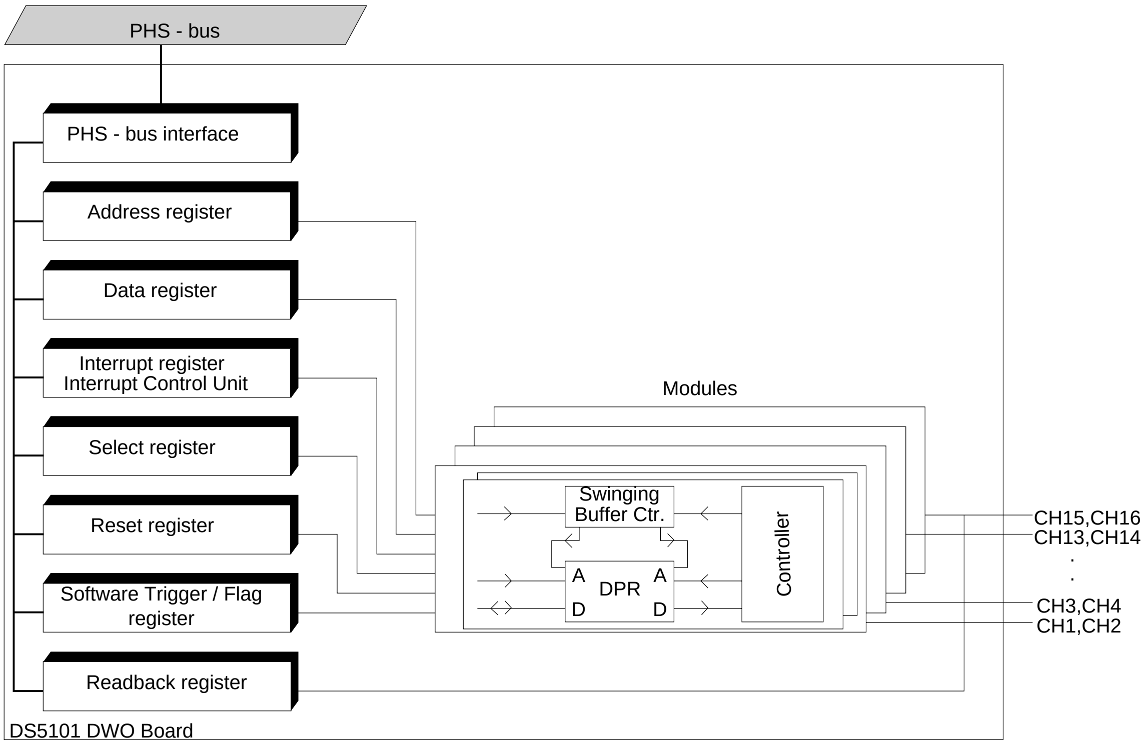


Figure 1.1. Block diagram of the DS5101.

The Digital Waveform Output board is designed modular consisting of a motherboard (DS5101) and up to eight modules (DS5101M). The motherboard contains the PHS-bus interface, I/O connector, clock distribution and eight module sockets. The modules (DS5101M) contain two signal generation channels each consisting of a controller and a local RAM.
From version DS5101-04 on, all eight modules have been integrated into the baseboard logic.

2 Getting Started

This section is intended to describe all the necessary steps to implement a DWO application on the DS5101 and how to control individual parameters by the DSP. Implementation of a custom specific DWO application as well as using one of the standard DWO applications that come with the DS5101 Software Environment is demonstrated in the following subsections.

The implementation of a custom specific DWO application starts with writing of the DWO program itself. Compilation of the DWO application source file (*.src*) by using the DWO compiler DWOCOMP results in a C source file (*.c*) and a related header file (*.h*). The C file contains the encoded DWO object code and a load routine for downloading the object code to the DS5101 by a master DSP. It can also contain ready-to-use update functions to modify delay values while the application is running. The header file contains address definitions for all timing parameters declared in the DWO source file. These definitions allow easy access to individual timing parameters by the master DSP.

The next step is writing the master DSP program or augmenting an existing DSP program by the DWO load and access functions. The DSP program must load the DWO application by using the load routine generated by DWOCOMP. It may then supply initial timing parameter values in case the values specified in the DWO source file are not suitable. A start function allows to start the signal generation on individual DWO channels. Timing parameters can be modified at any time without interrupting the signal generation.

2.1 A simple DWO Application

The DWO program listed below shows the implementation of a simple PWM generator. It may serve as an example for how to write your own first DWO applications. Two DWO channels are used in order to demonstrate interaction between different channels. A PWM generator can of course also be implemented on a single channel.

```
# ==========================================================================
#  PWM Generation
#
#  DS5101 DWO-compiler example source file
#
#  (C) 1996, dSPACE GmbH
#
#  $RCSfile$ $Revision$ $Date$
# ==========================================================================

unit ticks                        # all delays given in 25ns ticks

# --------------------------------------------------------------------------
#  PWM master clock
# --------------------------------------------------------------------------

ch1

delay t0 2000                     # half-period (50usec)

  reset, t0;                      # low after reset, delay t0
begin:
  syncen,                         # update parameters
    tr2,                          # trigger channel 2
    set, t0;                      # set output high, delay t0
  reset, t0,                      # set output low, delay t0
    goto begin;                   # loop
```

```
# -----------------------------------------------------------------------
#  PWM output
# -----------------------------------------------------------------------

ch2

delay t1 2000                   # duty-cycle (50usec / 50%)

  reset, wait;                  # low after reset, wait for trigger
begin:
  syncen,                       # update parameters
    set, t1;                    # set output high, delay t1
  reset, wait,                  # set output low, wait for trigger
    goto begin;                 # loop


updfunc all                     # generate update function 'all'
 1 t0,                          # which modifies t0 from channel 1
 2 t1;                          # and t1 from channel 2
```

Channel 1 generates the PWM master clock. The PWM period depends on the time delay *t0*, representing the half PWM period. The delay *t0* is initially set to 50 µs and thus yields an initial PWM period of 100 µs. With each rising edge of the master clock a trigger is sent to channel 2 (command *tr2*). A new time delay supplied by the DSP is accepted with the next rising edge, i.e. at the beginning of the next PWM period. This is controlled by the *syncen* command.

Channel 2, generating the actual PWM signal, is triggered with each rising edge of the master clock. The time delay *t1* yields the PWM duty-cycle duration. A new time delay is also accepted at the beginning of the next PWM period.

To compile the DWO source file type

```
dwocomp pwm1
```

The DWO compiler will generate the following files:

pwm1.c	encoded DWO application and C load routine
pwm1.h	time delay addresses, channel mask
pwm1.lst	listing file
pwm1.dat	DWO simulator input file

2.2 The DSP Program

The DSP program loading the PWM application to the DS5101 and controlling the PWM parameters might read as follows.

```
/*************************************************************************
*
* FILES:
*   pwm1_5101_hc.C
*
* DESCRIPTION:
*   PWM example application for dSPACE DS5101 DWO board.
*   The duty-cycle is periodically varied from 0.0 to 1.0.
*
* USAGE:
*   Connect DS5101 output channels 1 and 2 to an oscilloscope.
*
* VERSIONINFO:
*   VERSION:      1.1
*   RELEASE DATE: 03.09.97
*   AUTHOR(S):    H.-J. Miks, F. Beny
*                 dSPACE GmbH, Technologiepark 25, 33100 Paderborn
*
* $RCSfile:$ $Revision:$ $Date:$
*************************************************************************/

#include <brtenv.h>                       /* basic real time environment */
#include <ds5101.h>                           /* DS5101 access functions */
#include "pwm1.h"                         /* DWO application header file */
#include "pwm1.c"                        /* DWO application load routine */

float prd    = 10.0e-6;                                /* PWM period (sec) */
float duty   = 0.1;                                      /* PWM duty cycle */
float d_duty = 0.01;                               /* duty-cycle increment */

volatile int *error = (int *) (DP_MEM_BASE+DP_MEM_SIZE-1);


void isr_t0()                        /* timer0 interrupt service routine */
{
  duty += d_duty;                                  /* increment duty cycle */
  if (duty >= 1.0) duty -= 1.0;                 /* map to range 0.0..1.0 */

  ds5101_pwm1_update_all(                             /* update duty cycle */
    DS5101_1_BASE,
    prd * 0.5,                                         /* t0 = half-period */
    prd * duty);                                        /* t1 = duty cycle */
}
```

```
main()
{
  init();

  ds5101_init(DS5101_1_BASE);           /* basic DS5101 initialization */

  msg_info_set(MSG_SM_RTLIB, 0, "System started.");

  ds5101_pwm1_load(                            /* load DWO application */
    DS5101_1_BASE,
    DS5101_PWM1_CHANNELS);

  ds5101_pwm1_update_all(                  /* setup initial PWM values */
    DS5101_1_BASE,
    prd * 0.5,                                     /* t0 = half-period */
    prd * duty);                                   /* t1 = duty cycle */

  ds5101_start(                                /* start PWM generation */
    DS5101_1_BASE,
    DS5101_PWM1_CHANNELS);

  isr_t0_start(0.01);              /* start timer interrupt generation */

  while(1);                                     /* wait for interrupts */
}
```

After the basic initialization of the DS5101 the PWM application is loaded by using the function *ds5101_pwm1_load()*. This function is contained in the file *pwm1.c* generated by the DWO compiler. A channel mask parameter allows to load individual DS5101 channels without affecting the remaining channels. Here the channel mask *DS5101_PWM1_CHANNELS* from the header file *pwm1.h* is used to load those channels actually used by the PWM application (i.e. channels 1 and 2).

After a successful DWO application download the initial PWM period and duty-cycle parameters are set to 10 µs and 0.1, respectively, by using the DS5101 update function *ds5101_pwm1_update_all()* which was generated by DWOCOMP. This overwrites the default time delays specified in the DWO application source file.

The function *ds5101_start()* starts the PWM generation. The channel mask *DS5101_PWM1_CHANNELS* again allows to start the channels used by the application without affecting other channels.

You may comment out the call to *start_isr_t0()* and compile/download the program by typing

```
down1003 pwm1_5101_hc.c
```

The DS5101 will start to generate a square-wave signal of 100 µs period on channel 1 and a PWM signal of 100 µs period and 10 % duty-cycle on channel 2.

Now reactivate *isr_t0_start()* and again compile/download the program. The duty-cycle will now periodically rise from 0.0 to 1.0. The duty-cycle is updated at a constant rate of 100 Hz in the timer interrupt service routine.

2.3 Using a Standard DWO Application

Several standard DWO applications and corresponding parameter update functions are distributed with the DS5101 Software Environment (cf. section 3.1). This section describes how to implement a PWM generation by using the standard PWM application *\dspace\ds5101\pwm.src*. Deviating from the PWM generator described in the foregoing sections this one requires only a single channel for generation of a PWM signal. This DWO application is described in detail in section 3.1.3.

In order to implement PWM generators on individual DS5101 channels, the file *pwm.src* contains the source code for 16 uniform PWM generators. The channel mask parameter of the load and start functions selects the channels being actually loaded or started, respectively. The channel mask is generated by the macro *AP5101_PWM_MASK()*.

The DSP program now reads as shown below. An excerpt of the standard DWO application is listed in section 3.1.3.

```
/***********************************************************************
*
* FILES:
*   pwm2_5101_hc.C
*
* DESCRIPTION:
*   PWM example application for dSPACE DS5101 DWO board
*   using a standard DWO application.
*   The duty-cycle is periodically varied from 0.0 to 1.0.
*
* USAGE:
*   Connect DS5101 output channel 1 to an oscilloscope.
*
* VERSIONINFO:
*   VERSION:      1.1
*   RELEASE DATE: 03.09.97
*   AUTHOR(S):    H.-J. Miks, F. Beny
*                 dSPACE GmbH, Technologiepark 25, 33100 Paderborn
*
* $RCSfile:$ $Revision:$ $Date:$
***********************************************************************/
```

```
#include <brtenv.h>                      /* basic real-time environment */
#include <ds5101.h>                          /* DS5101 access functions */
#include <ap5101.h>   /* DS5101 standard application support functions */

float prd    = 10.0e-6;                              /* PWM period (sec) */
float duty   = 0.1;                                  /* PWM duty-cycle */
float d_duty = 0.01;                           /* duty-cycle increment */

void isr_t0()                      /* timer0 interrupt service routine */
{
  duty += d_duty;                              /* increment duty-cycle */
  if (duty >= 1.0) duty -= 1.0;               /* map to range 0.0..1.0 */

  ds5101_pwm_update(                   /* update period and duty-cycle */
    DS5101_1_BASE,
    1, prd, duty);
}


main()
{
  init();

  ds5101_init(DS5101_1_BASE);          /* basic DS5101 initialization */

  msg_info_set(MSG_SM_RTLIB, 0, "System started.");

  ds5101_pwm_load(                             /* load DWO application */
    DS5101_1_BASE, AP5101_PWM_MASK(1));

  ds5101_pwm_update(        /* setup initial PWM period and duty-cycle */
    DS5101_1_BASE,
    1, prd, duty);

  ds5101_start(                                /* start PWM generation */
    DS5101_1_BASE, AP5101_PWM_MASK(1));

  isr_t0_start(0.01);              /* start timer interrupt generation */

  while(1);                                     /* wait for interrupts */
}
```

3 DS5101 Software Environment

3.1 Standard DWO Applications

Some standard DWO applications and related functions are delivered with the DS5101 Software Environment. They can be used directly or may serve as templates to write your own individual DWO application programs.

All the applications described in the following subsections are implemented in multiple instances (blocks) using different DS5101 channels. The number of available blocks of a particular application depends on the number of output channels required. The blocks being actually loaded can be selected by the channel mask parameter of the corresponding load function. Thus the DS5101 may execute different applications using different output channels at the same time.

Be sure to specifiy the appropriate load mask, otherwise the application will not operate properly. The header file *ap5101.h* contains macros for the different applications returning the channel mask for a given block number.

```
AP5101_INC_MASK(blk)
AP5101_MONO_MASK(blk)
AP5101_PWM_MASK(blk)
AP5101_PWM3_MASK(blk)
AP5101_PWM6_MASK(blk)
```

All time delays declared within the DWO application source files are preset to a value of zero. Suitable values must be supplied by using the appropriate parameter update function before execution of a DWO application is started (cf. the example program listed in section 2.3).

3.1.1 Incremental Sensor Simulation

This demo application provides up to 7 independent incremental sensor blocks with leading and lagging phase outputs each (phi0, phi90). The maximum number of blocks is limited by the 7 control flags available for direction control. Table 3.3 shows the channel assignment, channel mask, and direction control flag assignment for the 7 incremental sensor blocks.

Block	Channels		Channel mask	Direction flag
	phi0	phi90		
1	1	2	0x0003	1
2	3	4	0x000C	2
3	5	6	0x0030	3
4	7	8	0x00C0	4
5	9	10	0x0300	5
6	11	12	0x0C00	6
7	13	14	0x3000	7

Table 3.3. Incremental sensor blocks

The DWO source code of a single incremental sensor is listed below. The program is based on the time constant *ts*. It is initially set to 0 and must be set to a quarter of the initial signal period before the program is started.

The four program states *l1 .. l4*, each representing a quarter of one signal period, are passed in positive or negative direction depending on the state of the direction control flag.

Immediate update mode is used, because only a single parameter must be updated.

```
delay ts 0                      # time per section (period/4)

# --------------------------------------------------------------------------
#  incremental sensor block 1 (channels 1, 2)
# --------------------------------------------------------------------------

ch1                             # phi0
immediate                       # select immediate update mode
l1:                             # section 1
  if flag1                      # positive direction
    set, ts,  goto l2,          # output high, wait ts, step forward
  else                          # negative direction
    set, ts,  goto l4;          # output high, wait ts, step backward
l2:                             # section 2
  if flag1                      # positive direction
    set, ts,  goto l3,          # output high, wait ts, step forward
  else                          # negative direction
    set, ts,  goto l1;          # output high, wait ts, step backward
l3:                             # section 3
  if flag1                      # positive direction
    reset, ts,  goto l4,        # output low, wait ts, step forward
  else                          # negative direction
    reset, ts,  goto l2;        # output low, wait ts, step backward
l4:                             # section 4
  if flag1                      # positive direction
    reset, ts,  goto l1,        # output low, wait ts, step forward
  else                          # negative direction
    reset, ts,  goto l3;        # output low, wait ts, step backward

ch2                             # phi90
immediate                       # select immediate update mode
l1:                             # section 1
  if flag1                      # positive direction
    reset, ts,  goto l2,        # output low, wait ts, step forward
  else                          # negative direction
    reset, ts,  goto l4;        # output low, wait ts, step backward
l2:                             # section 2
  if flag1                      # positive direction
    set, ts,  goto l3,          # output high, wait ts, step forward
  else                          # negative direction
    set, ts,  goto l1;          # output high, wait ts, step backward
l3:                             # section 3
  if flag1                      # positive direction
    set, ts,  goto l4,          # output high, wait ts, step forward
  else                          # negative direction
    set, ts,  goto l2;          # output high, wait ts, step backward
l4:                             # section 4
  if flag1                      # positive direction
    reset, ts,  goto l1,        # output low, wait ts, step forward
  else                          # negative direction
    reset, ts,  goto l3;        # output low, wait ts, step backward
```

3.1.2 Mono-Flop

The mono-flop application example also uses 2 output channels for each of the 8 available mono-flop blocks, an inverted (/y) and a non-inverted output (y). The channel assignment for the different blocks is given in table 3.4.

Block	Channels		Channel mask
	y	/y	
1	1	2	0x0003
2	3	4	0x000C
3	5	6	0x0030
4	7	8	0x00C0
5	9	10	0x0300
6	11	12	0x0C00
7	13	14	0x3000
8	15	16	0xC000

Table 3.4. Mono-flop blocks

The following listing shows the DWO source code of a single mono-flop block. Operation is very simple. After having received a trigger event the respective output is set or reset, respectively, for the duration given by the parameter *tm*. Trigger events may be received from on-board trigger sources (other channels), or from the master DSP. Again immediate update mode is used.

```
delay tm 0                       # t_mono

# -------------------------------------------------------------------------
#  mono-flop 1 (channels 1, 2)
# -------------------------------------------------------------------------

ch1                              # non-inverted output
immediate                        # select immediate update mode
  reset, wait;                   # low after reset, wait for trigger
begin:
  set, tm;                       # set output high, wait tm
  reset, wait,                   # set output low, wait for trigger
    goto begin;                  # loop

ch2                              # inverted output
immediate                        # select immediate update mode
  set, wait;                     # high after reset, wait for trigger
begin:
  reset, tm;                     # set output low, wait tm
  set, wait,                     # set output high, wait for trigger
    goto begin;                  # loop
```

3.1.3 PWM Generation

Generation of a simple PWM signal is one of the standard examples. Each PWM signal requires a single output channel, thus resulting in an overall amount of 16 available PWM signals.

Block	Channel	Channel mask
1	1	0x0001
2	2	0x0002
3	3	0x0004
4	4	0x0008
5	5	0x0010
6	6	0x0020
7	7	0x0040
8	8	0x0080

Block	Channel	Channel mask
9	9	0x0100
10	10	0x0200
11	11	0x0400
12	12	0x0800
13	13	0x1000
14	14	0x2000
15	15	0x4000
16	16	0x8000

Table 3.5. 1-phase PWM blocks

The high- and low-level durations can be adjusted independently through the parameters *t1* and *t2*. The sum of *t1* and *t2* yields the PWM period. Because both time parameters depend on each other, synchronous update mode must be used.

```
delay t1 0                      # t_high
delay t2 0                      # t_low

# ------------------------------------------------------------------------
#  PWM channel 1
# ------------------------------------------------------------------------

ch1
begin:
  syncen,                       # use new data from DSP
    set, t1;                    # set output high, wait t1
  reset, t2,                    # set output low, wait t2
    goto begin;                 # loop
```

3.1.4 3-Phase PWM Generation

The 3-phase PWM application generates ABC-phase PWM signals symmetrical to the middle of their high level pulses. To synchronize the 3 output signals a master clock is generated and thus 4 channels are required for each PWM block.

Block	Channels				Channel mask
	A	B	C	clock	
1	1	2	3	4	0x000F
2	5	6	7	8	0x00F0
3	9	10	11	12	0x0F00
4	13	14	15	16	0xF000

Table 3.6. 3-phase PWM blocks

The high- and low-level durations can be adjusted independently through the parameters *ta1* and *ta2, tb1* and *tb2,* or *tc1* and *tc2* for the respective PWM phase. The sum of 2 *tx1* and *tx2* yields the PWM period (see figure 4.1. below). All 3 PWM periods resulting from the respective delay parameters must match the master clock period 2 *t0*. Therefore it is recommended to use the function *ds5101_pwm3_update()* for parameter update.

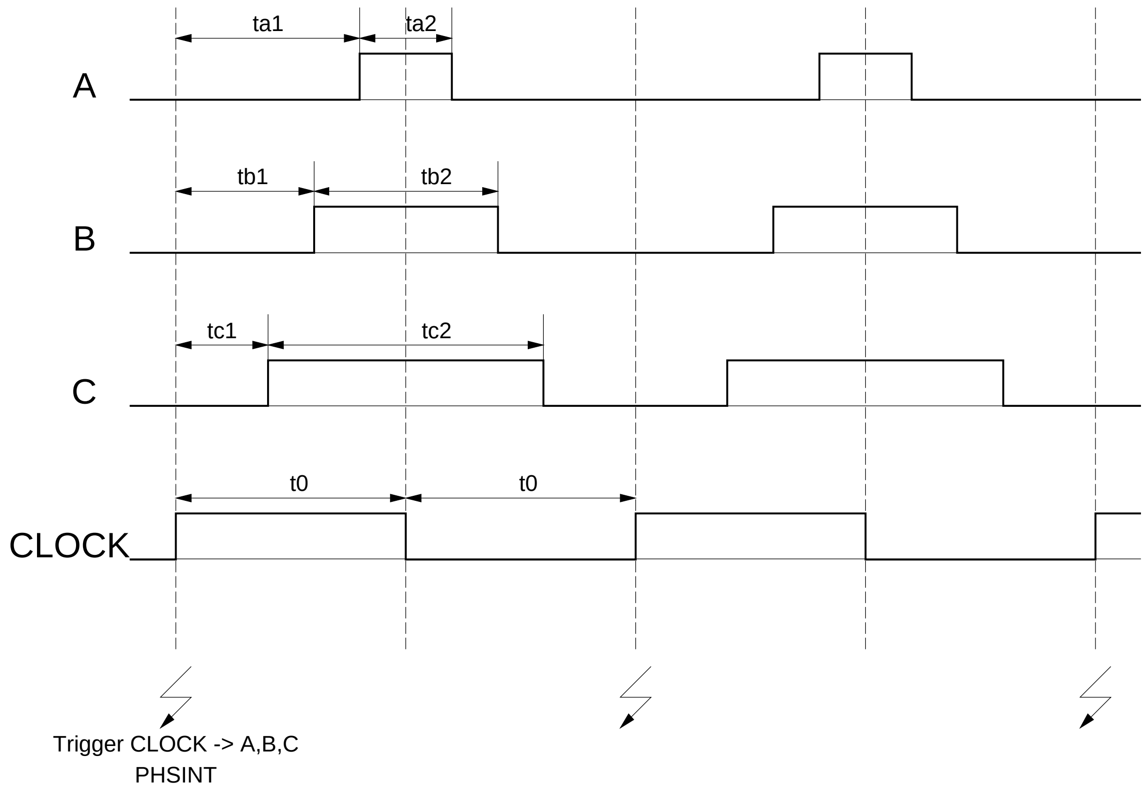


Figure 3.1. 3 Phase PWM Generation

```
# ----------------------------------------------------------------------
#  PWM block 1 (channels 1..4)
# ----------------------------------------------------------------------

ch1                                # phase A
delay ta1 0                        # t_low/2, phase A
delay ta2 0                        # t_high,  phase A

  reset, wait;                     # low after reset, wait for trigger
begin:
  syncen,                          # use new data from DSP
    nop, ta1;                      # delay t_low/2
  set, ta2;                        # set output high, delay t_high
  reset, wait,                     # set output low, wait for trigger
    goto begin;                    # loop
```

```
ch2                                # phase B
delay tb1 0                        # t_low/2, phase B
delay tb2 0                        # t_high,  phase B

  reset, wait;                     # low after reset, wait for trigger
begin:
  syncen,                          # use new data from DSP
    nop, tb1;                      # delay t_low/2
  set, tb2;                        # set output high, delay t_high
  reset, wait,                     # set output low, wait for trigger
    goto begin;                    # loop

ch3                                # phase C
delay tc1 0                        # t_low/2, phase C
delay tc2 0                        # t_high,  phase C

  reset, wait;                     # low after reset, wait for trigger
begin:
  syncen,                          # use new data from DSP
    nop, tc1;                      # delay t_low/2
  set, tc2;                        # set output high, delay t_high
  reset, wait,                     # set output low, wait for trigger
    goto begin;                    # loop

ch4                                # master clock
delay t0  0                        # PWM period / 2


  reset, t0;                       # low after reset
begin:
  syncen,                          # use new data from DSP
    phsint,                        # send interrupt to DSP
    tr1, tr2, tr3,                 # trigger channels 1..3
    set, t0;                       # set output high, delay t0
  reset, t0,                       # set output low, delay t0
    goto begin;                    # loop
```

3.1.5 3-Phase PWM with inverted and non-inverted Outputs

This 3-phase PWM application generates inverted and non-inverted ABC-phase PWM signals symmetrical to the middle of their high level pulses. To synchronize the 6 output signals a master clock is generated and thus 7 channels are required for each PWM block.

Block	Channels							Channel mask
	A	/A	B	/B	C	/C	clock	
1	1	2	3	4	5	6	7	0x007F
2	9	10	11	12	13	14	15	0x7F00

Table 3.7. 3-phase PWM blocks with low and high outputs

The high- and low-level durations can be adjusted independently through the parameters *ta1 .. ta4*, *tb1 .. tb4*, or *tc1 .. tc4* for the respective PWM phase. A switch delay td between the correspinding edges of the inverted and related non-inverted phase output is achieved by adjusting the respective time parameters appropriately (see figure 4.2.). In addition all PWM periods resulting from the respective delay parameters must match the master clock period 2 *t0* (cf. the DWO source code listed below). Therefore it is recommended to use the function *ds5101_pwm6_update()* for parameter update.

For 3-phase PWM generation with inverted and non-inverted outputs (PWM6) please note the following limitations. Generation of PWM signals with duty cycle values of 0 (non-inverted output constant low) and 1 (non-inverted output constant high) is not possible. As a consequence a remaining high pulse of 250 ns is generated at the non-inverted output for duty cycle values near or equal to 0. For duty cycle values near or equal to 1 the inverted output generates a high pulse of 800 ns. The actual duty cycle limits depend on the PWM period tp and the switch delay td.
Please note that the function ds5101_pwm6_update() does not perform any parameter range checks and negative values for tp or td, or duty cycle values outside the valid range 0 <= duty <= 1 will produce unpredictable results.

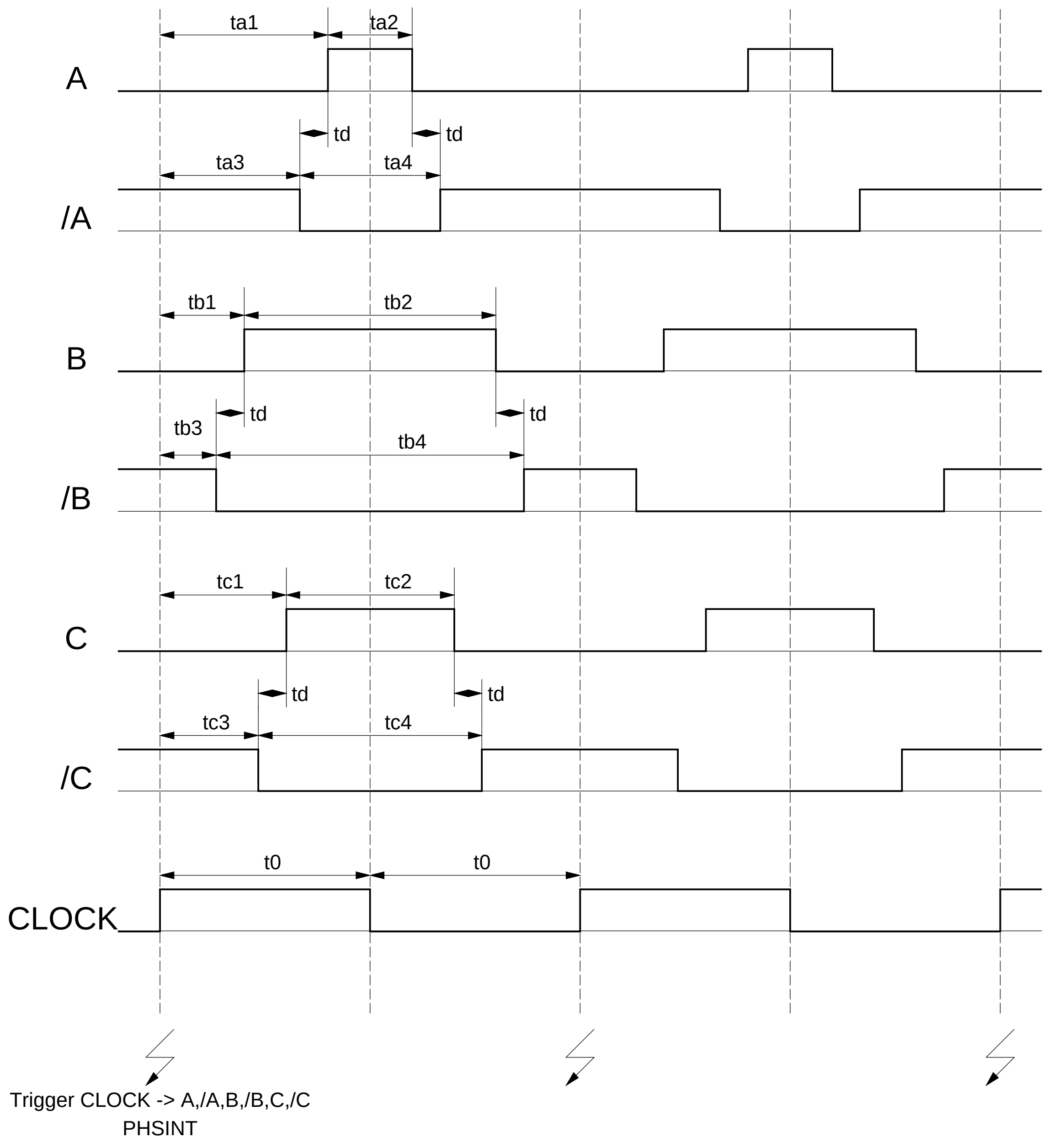


Figure 3.2. 3 Phase PWM Generation with inverted Outputs

```
# --------------------------------------------------------------------------
#  PWM block 1 (channels 1..7)
# --------------------------------------------------------------------------

ch1                               # phase A
delay ta1 0                       # t_low/2,  phase A
delay ta2 0                       # t_high,   phase A

  reset, wait;                    # low after reset, wait for trigger
begin:
  syncen,                         # use new data from DSP
    nop, ta1;                     # delay t_low/2
  set, ta2;                       # set output high, delay t_high
  reset, wait,                    # set output low, wait for trigger
    goto begin;                   # loop

ch2                               # phase /A
delay ta3 0                       # t_high/2, phase /A
delay ta4 0                       # t_low,    phase /A
  set, wait;                      # high after reset, wait for trigger
begin:
  syncen,                         # use new data from DSP
    nop, ta3;                     # delay t_high/2
  reset, ta4;                     # set output low, delay t_low
  set, wait,                      # set output high, wait for trigger
    goto begin;                   # loop

ch3                               # phase B
delay tb1 0                       # t_low/2,  phase B
delay tb2 0                       # t_high,   phase B

  reset, wait;                    # low after reset, wait for trigger
begin:
  syncen,                         # use new data from DSP
    nop, tb1;                     # delay t_low/2
  set, tb2;                       # set output high, delay t_high
  reset, wait,                    # set output low, wait for trigger
    goto begin;                   # loop

ch4                               # phase /B
delay tb3 0                       # t_high/2, phase /B
delay tb4 0                       # t_low,    phase /B

  set, wait;                      # high after reset, wait for trigger
begin:
  syncen,                         # use new data from DSP
    nop, tb3;                     # delay t_high/2
  reset, tb4;                     # set output low, delay t_low
  set, wait,                      # set output high, wait for trigger
    goto begin;                   # loop
```

```
ch5                               # phase C
delay tc1 0                       # t_low/2,  phase C
delay tc2 0                       # t_high,   phase C

  reset, wait;                    # low after reset, wait for trigger
begin:
  syncen,                         # use new data from DSP
    nop, tc1;                     # delay t_low/2
  set, tc2;                       # set output high, delay t_high
  reset, wait,                    # set output low, wait for trigger
    goto begin;                   # loop

ch6                               # phase /C
delay tc3 0                       # t_high/2, phase /C
delay tc4 0                       # t_low,    phase /C

  set, wait;                      # high after reset, wait for trigger
begin:
  syncen,                         # use new data from DSP
    nop, tc3;                     # delay t_high/2
  reset, tc4;                     # set output low, delay t_low
  set, wait,                      # set output high, wait for trigger
    goto begin;                   # loop

ch7                               # master clock
delay t0  0                       # PWM period / 2


  reset, t0;                      # low after reset
begin:
  syncen,                         # use new data from DSP
    phsint,                       # send interrupt to DSP
    tr1, tr2, tr3,                # trigger channels 1..6
    tr4, tr5, tr6,
    set, t0;                      # set output high, delay t0
  reset, t0,                      # set output low, delay t0
    goto begin;                   # loop
```

4 Compiler DWOCOMP

4.1 Preface

The controller of every DS5101 channel executes a program which is stored in its local RAM. Every state needs up to 128 bits of code. To spare the user the task of coding all those bits, this compiler has been written.

The user can describe the timing and dependencies of all output signals in a text based source file. Every command can contain several different operations, like
- set the output pin to the desired value
- start a timer with a given time constant and wait until the time has passed
- trigger one or several other channels
- generate an interrupt at the PHS bus
- do a conditional branch

Commands for the simulator tool can be added to the source file.

DWOCOMP translates this source file (**.src*) into a C source code file (**.c*), which can be linked to C based application programs.Results and error messages are reported in a list file (**.lst*).

This documentation describes the use of the compiler, the source file syntax, compiler and simulator error messages and output file formats.

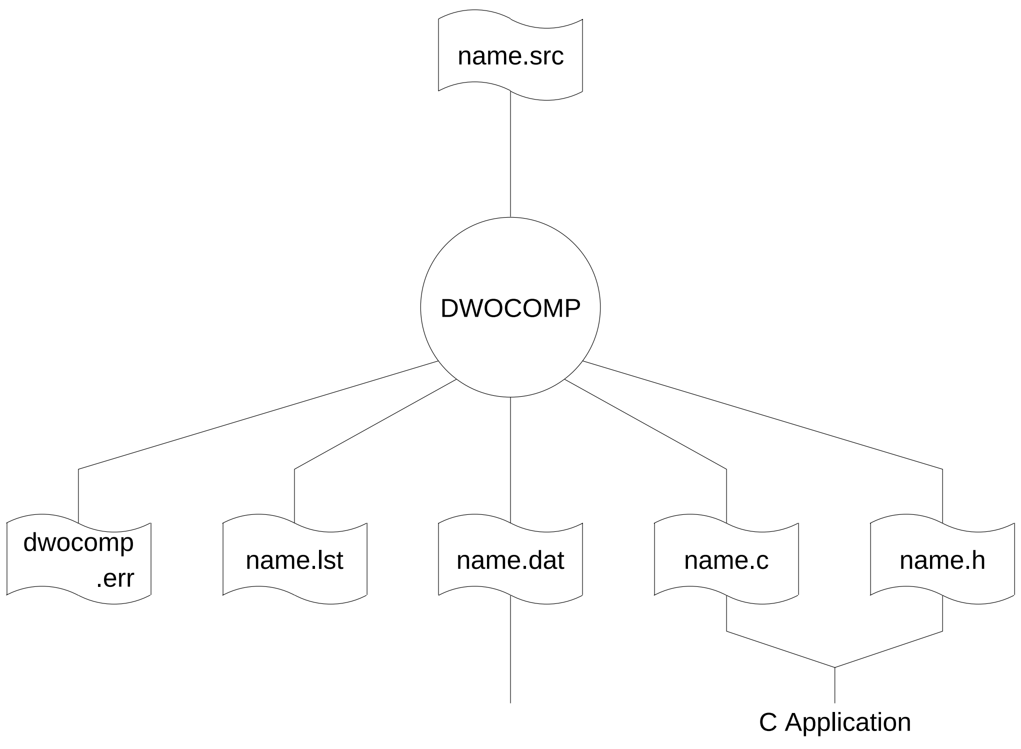


Figure 4.1. Data flow between DWOCOMP and C Application

4.2 Compiler Description

4.2.1 Syntax

4.2.1.1 General

4.2.1.1.1 Command line

DWOCOMP is started with the command

```
>DWOCOMP filename [options]
```

where filename is the name of the source file without the extension *.src*. This extension is mandatory.

Options can be

-m-	messages off
-m+	messages on (default)
-e-	no error file (default)
-e+	generate error file
-n (name)	use (name) in C source and header files as program name

4.2.1.1.2 Options

-m-

All screen messages (pass number, line numbers, errors and warnings) will be supressed. This can be useful, if the compiler is called from within another program or from a 'DOS box' in WINDOWS (r). Errors and warnings are written to the listfile and, if enabled by option -e+, to the error file *dwocomp.err*.

-m+

All screen messages (pass number, line numbers, errors and warnings) will be written to the screen (default mode).

-e-

No error file *dwocomp.err* will be generated. Errors and warnings will be written to the list file (default mode).

-e+

First an existing error file *dwocomp.err* will be deleted. If any errors or warnings occur, an error file will be opened, and all errors and warnings will be written to this file and to the list file.

-n (name)

(name) will be used in the C source code and header files as program name. If no '-n' command is given, the source file name will be used.
Example:

```
  dwocomp test -n demo
```

will translate *test.src* to *test.c*, *test.h*, *test.lst* and *test.dat*. In *test.c*, however, the load function will be named *ds5101_demo_load* instead of *ds5101_test_load*.

4.2.1.1.3 Case Sensitivity

DWOCOMP is not case sensitive. The set command may be written as *set*, *SET*, *Set* or any other case combination.

4.2.1.1.4 Separators

Commands may be separated by one of the following characters:

- " "	(blank)
- ,	(comma)
- tab	(tabulator)
- "="	(equal sign)
- CR	(carriage return)
- LF	(line feed)

or by a comment (beginning with "#").

Do not use the ";" character (end of state) or the ":" character (*label*) as a separator!

4.2.1.1.5 Comments

Comments start with the hash symbol ("#"). Everything from this character to the end of the current line is interpreted as comment and ignored by the compiler.

4.2.1.1.6 Number Formats

Real numbers for delay time constants and simulation times may be entered by using one of the following formats:

1
1.
1.0
1e0
1.00e0

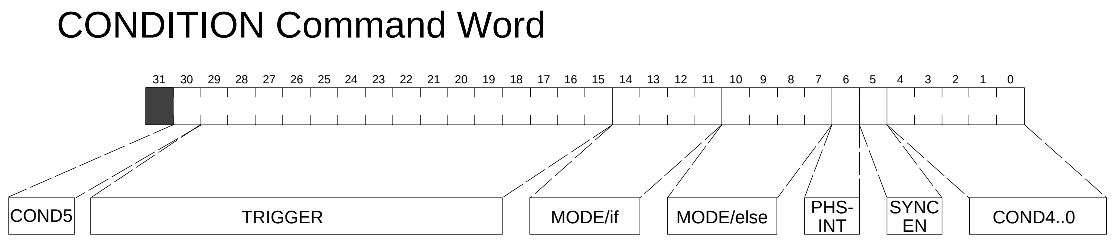
Integer numbers may be entered as

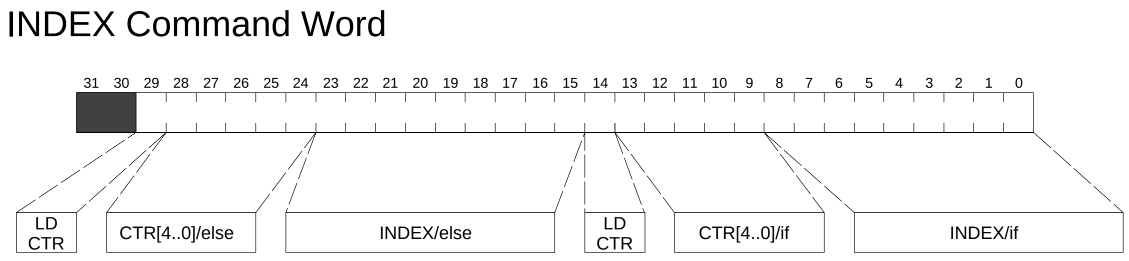
100	(decimal)
100d	(decimal)
64h	(hexadecimal, assembler style)
$64, $FF, $ff	(hexadecimal, PASCAL style)
$0064	(hexadecimal, PASCAL style)
0x64, 0xFF, 0xff	(hexadecimal, C style)
0x0064	(hexadecimal, C style)

4.2.1.1.7 RAM Areas

A state is translated into an opcode of 90 bits for the state commands and 30 bits for each delay constant.
Because the PHS bus interface of the DS5101 is 32 bit wide, the 90 bit state command is split into three words. The delay constant can be written directly. See section 6. for additional information.

The three state command words CONDITION, INDEX and NEXT are stored at consecutive addresses within the state buffer.





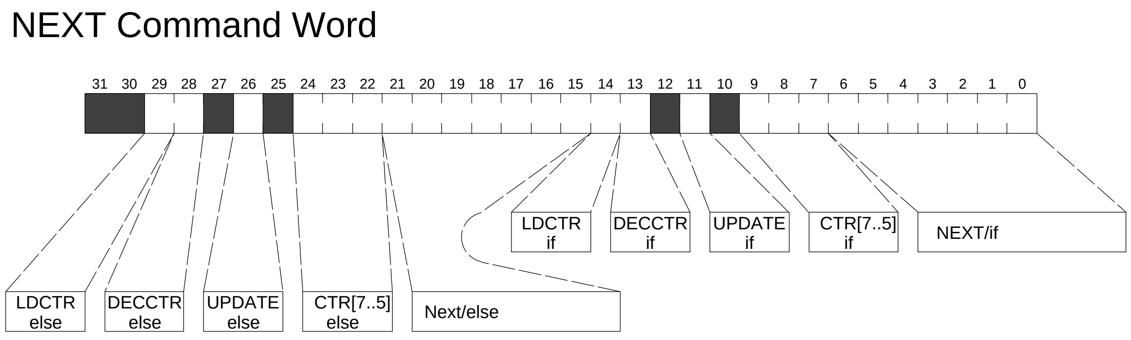


Figure 4.2. State Command Words

The delay constants DELAY are stored within the delay RAM. Each output channel has its own state and delay RAM.

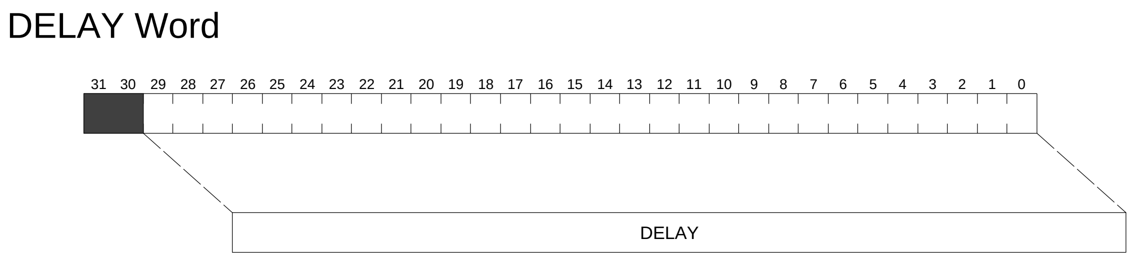


Figure 4.3. Delay Word

State and delay RAM are implemented in a dual port RAM (DPR) for each channel. The delay RAM is separated into three buffer areas, which serve as swinging buffers for consistent updating of delay constants during program execution.

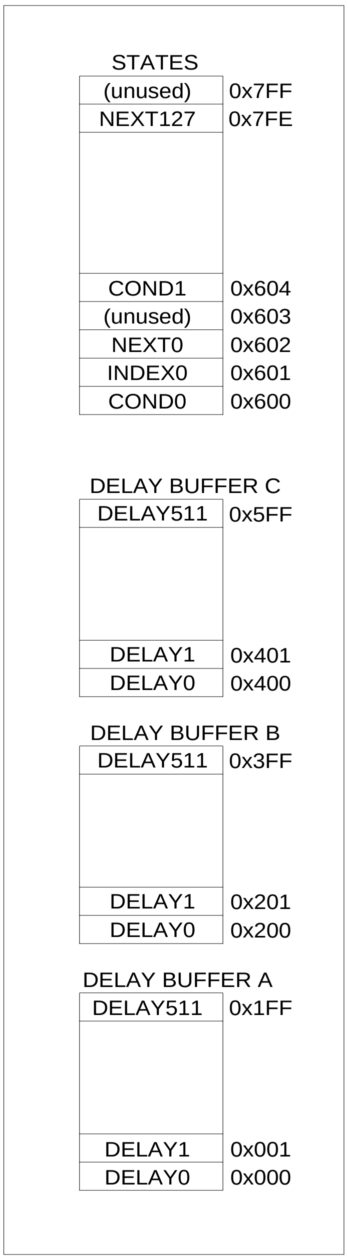


Figure 4.4. Dual Port Ram Address Map for one Channel

4.2.1.2 Syntax Graph

The syntax of the DWO compiler language is defined by this syntax graph:

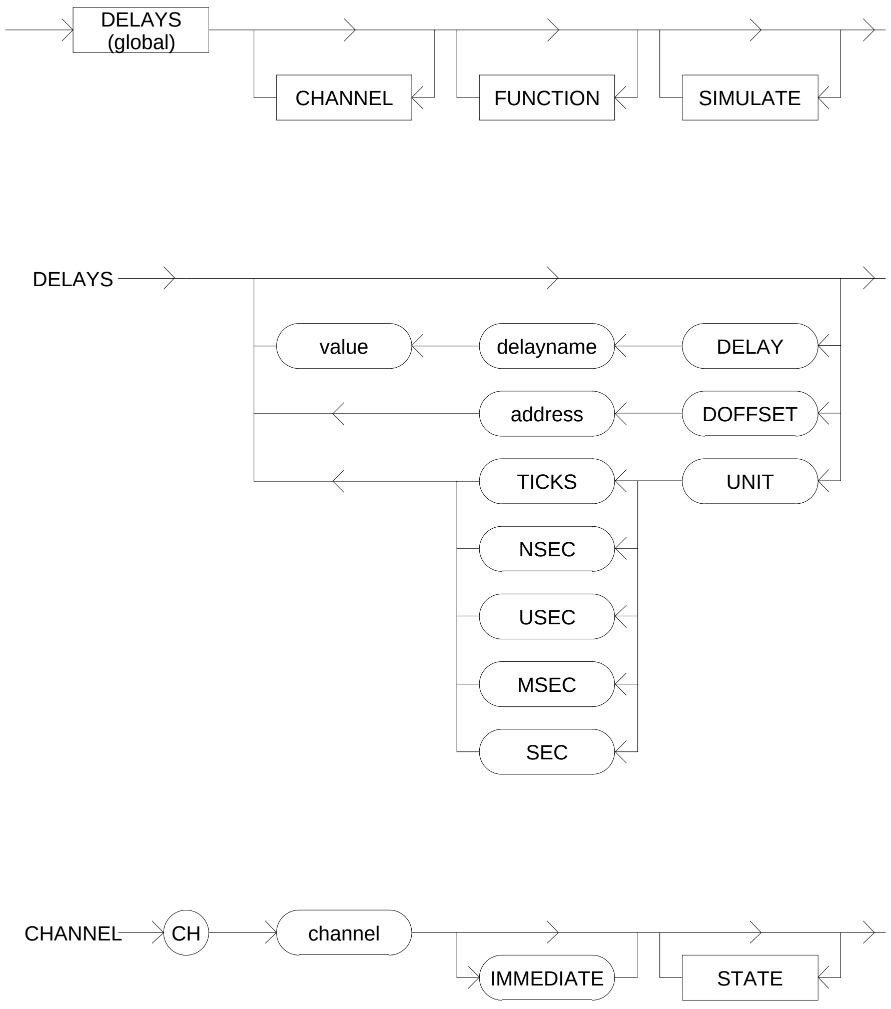


Figure 4.5. Syntax Graph - Part 1

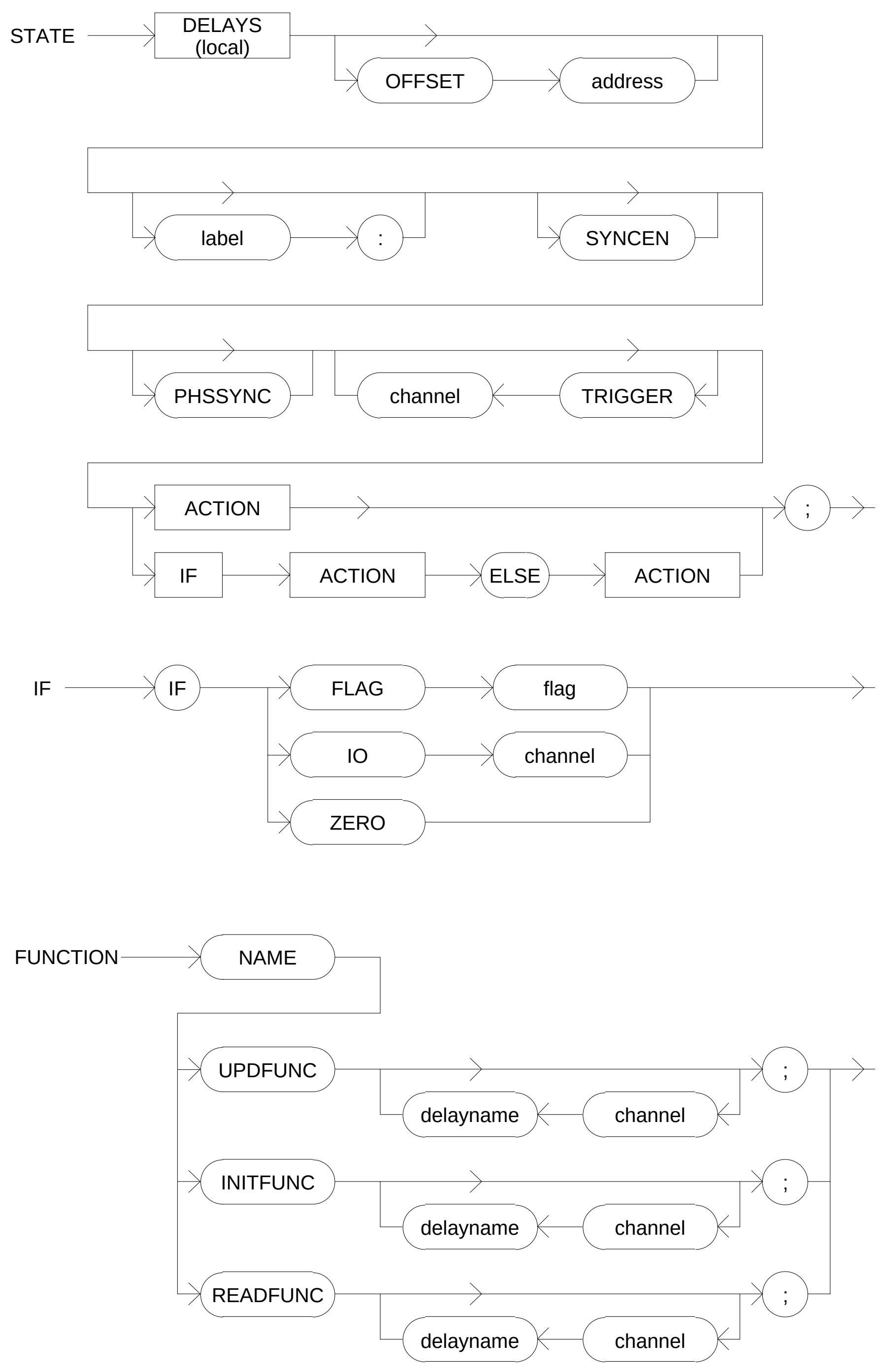


Figure 4.6. Syntax Graph - Part 2

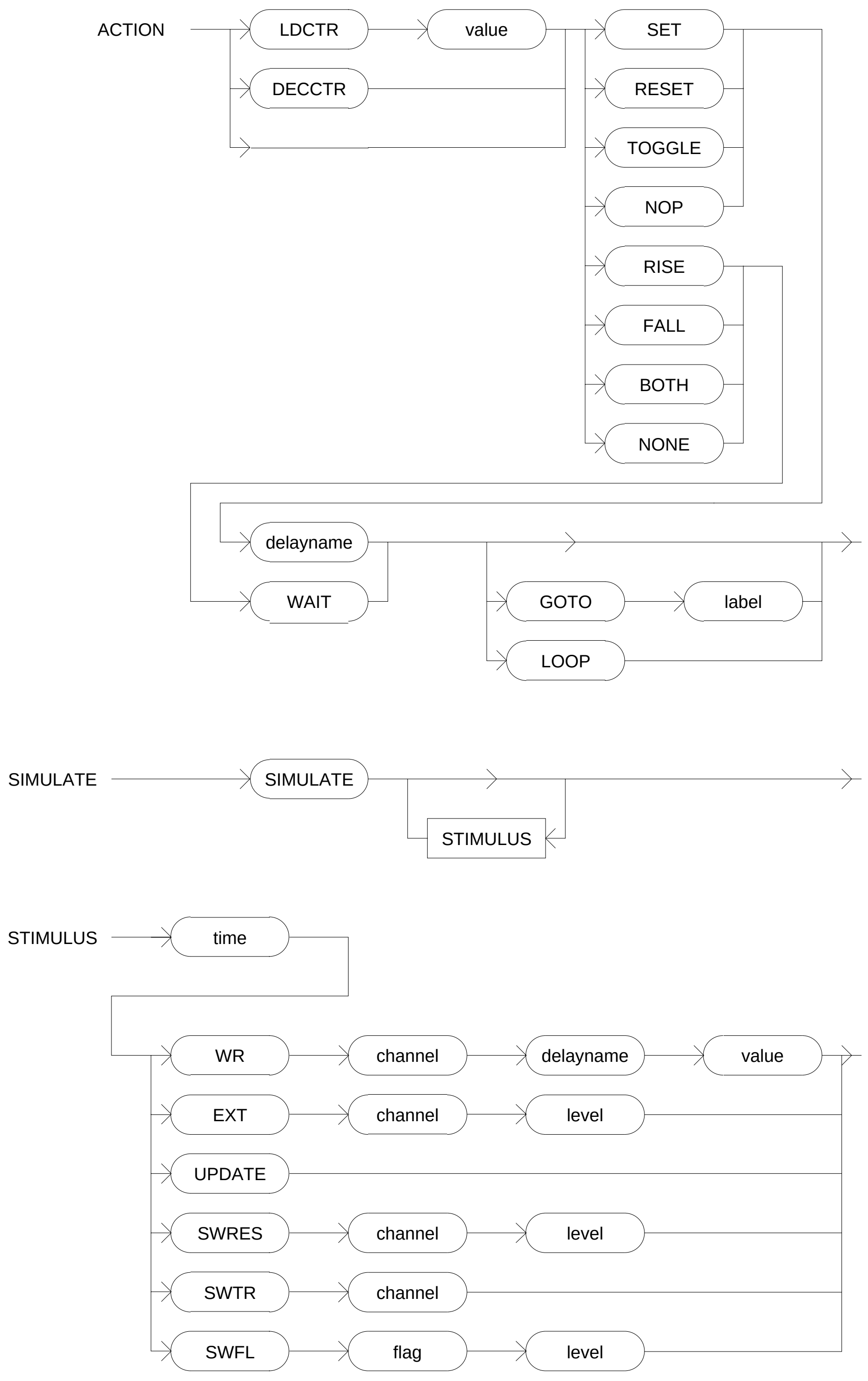


Figure 4.7. Syntax Graph - Part 3

4.2.1.3 Compiler Commands

The compiler commands are described as follows:

First the syntax of the command is given. If parameters can be written in several ways, all of them are listed. Example:

Syntax:	*CH n*
	CH = n
	CHn

Then the path in the syntax graph is shown, where the command can be found. Example: The command can be found in block ACTION. ACTION is a sub-block of block STATE. STATE is a sub-block of block CHANNEL, and CHANNEL can be found in the syntax graph root.

Syntax Graph: CHANNEL / STATE / ACTION

A command is translated to a bit pattern consisting of between 1 and 30 bits. This bit pattern is part of either CONDITION, INDEX, NEXT or delay command word. The position of the bit pattern for ACTION commands (see the syntax graph) within CONDITION, NEXT or INDEX command word changes, if the ACTION is part of an IF / ELSE construct. Therefore both bit pattern positions are given. Example:

RAM area: state RAM / CONDITION command word

Code:	D31..30	D29..15	D14	D13..11	D10	D9..7	D6	D5	D4..0
(IF)	-	-	0	011	-	-	-	-	-

Code:	D31..30	D29..15	D14	D13..11	D10	D9..7	D6	D5	D4..0
(ELSE)	-	-	-	-	0	011	-	-	-

A functional description of the command and an example follows.

Description:

The *BOTH* command causes the actual channel to stay in or change to input mode. A delay definition by *delayname* in this state is not needed and will be ignored. The channel will stay in this condition until the input level at the according pin changes from low to high or from high to low (both edges active). Then the next state of this channel will be executed.

Example:

```
both wait;       # stay in input mode until an edge is detected
```

4.2.1.3.1 BOTH

Syntax: *BOTH*

Syntax Graph: CHANNEL / STATE / ACTION

RAM area: state RAM / CONDITION command word

Code:	D31..30	D29..15	D14	D13..11	D10	D9..7	D6	D5	D4..0
(IF)	-	-	0	011	-	-	-	-	-

Code:	D31..30	D29..15	D14	D13..11	D10	D9..7	D6	D5	D4..0
(ELSE)	-	-	-	-	0	011	-	-	-

Description:

The *BOTH* command causes the actual channel to stay in or change to input mode. A delay definition by *delayname* in this state is not needed and will be ignored. The channel will stay in this condition until the input level at the according pin changes from low to high or from high to low (both edges active). Then the next state of this channel will be executed.

Example:

```
    both wait;       # stay in input mode until an edge is detected
```

4.2.1.3.2 CH (Channel)

Syntax:	*CHn* *CH n* *CH = n*
Syntax Graph:	CHANNEL
RAM area:	none (compiler directive)

Description:

CH defines the actual output channel for all states to follow. A separator between *CH* and *n* may be inserted.

Channel *n* must be in the range from 1 to 16.

CH may only be used once for an output channel. An attempt to define an already used output channel again will yield an error message.

Examples:

```
CH5              # start definition of channel 5
CH 5
CH=5
```

4.2.1.3.3 DECCTR

Syntax: *DECCTR*

Syntax Graph: CHANNEL / STATE / ACTION

RAM area: state RAM / NEXT command word

Code:	D31..14	D13	D12..9	D8..7	D6..0
(IF)	-	1	-	-	-

Code:	D31..29	D28	D27..24	D23..22	D21..0
(ELSE)	-	1	-	-	-

Description:

DECCTR decrements the 8 bit channel loop counter.

See the *LDCTR* command for further details.

Example:

```
    decctr, set t0;  # decrement counter, set output high for t0
```

4.2.1.3.4 DELAY

Syntax: *DELAY delayname value*

Syntax Graph: DELAYS
CHANNEL / STATE / DELAYS

RAM area: delay RAM

Code:

D31..D30	D29 .. D0
-	value

Description:

Delay defines a delay constant, which can be used in a state definition to define the time after which the next state will be executed.

All delay constants are stored in a separate delay buffer for each channel. They are addressed by using a symbolic delay name. Delays are only valid for the channel they had been defined for. However, delays declared before the first channel statement will be global and available for all channels, because those delays will be loaded to all channels by the download software.

Several states can use the same delay time by using the same *delayname*. The delay time itself must only be defined once in this case.

The *delayname* may be any string (without separators) of up to 20 characters.

Delays are stored at consecutive addresses in the delay RAM, starting at address 1 (address 0 is reserved for internal use). The delay address counter can be set by the *DOFFSET* command, if a delay has to be stored at a fixed address.

If different source files describing different channels shall use the same global delays, make sure to arrange them at the same delay addresses with the *DOFFSET* command.

The delay time value is multiplied by the compiler with the *UNIT* time base to obtain the number of 25 nsec machine cycles (*TICKS*). The number of *TICKS* must be between 0 and 0x3FFFFFFF. Therefore *value* has to be in the range
0 <= *value* < (0x3FFFFFFF * 25ns / *UNIT*).

Example:

If *UNIT* = sec, then
0 <= *value* < 26.84

Value must be given as a real number.

A value of less than 10 *TICKs* should be avoided, because the channel controller needs 10 clock cycles to execute a state. Values below 10 will be accepted, but execution will take 10 *TICKs*, and a warning is issued.

Examples:

```
delay T1 40       # Delay (T1) = 40 UNITs is stored at the next free delay
                  # RAM buffer address
doffset 100
delay T2 50       # Delay (T2) = 50 UNITs is stored at delay RAM address
                  # 100
```

4.2.1.3.5 Delayname

Syntax: *(delayname)*

Syntax Graph: CHANNEL / STATE / ACTION

RAM area: state RAM / INDEX command word

Code:	D31..28	D27..15	D14..9	D8..0
(IF)	-	-	-	delay address

Code:	D31..24	D23..15	D14..9	D8..0
(ELSE)	-	delay address	-	-

Description:

A *delayname* causes the channel controller to wait a certain time before proceeding to the next state. The channel has to be in output mode at this time.

The *delayname* must be defined before by using the *DELAY* command.

Different *delaynames* can be given for the *IF* and the *ELSE* action of a state.

Example: (*delayname* = ’t1’)

```
delay t1 1000    # define delay t1 = 1000 UNITs
set t1;          # set output high for 1000 UNITs
rise wait;       # enter input mode and wait for a rising edge
```

4.2.1.3.6 DOFFSET

Syntax: *DOFFSET address*

Syntax Graph: DELAYS
CHANNEL / STATE / DELAYS

RAM area: none (compiler directive)

Description:

DOFFSET causes the compiler to set its internal delay address counter for the delay buffers to a new value.

Address has to be in the range of 1 to 511. Address 0 is reserved for internal use.

Delays defined with the *DELAY* command will be stored at this address. The counter is incremented afterwards. If the delay RAM location is already in use, an error is reported.

The delay address counter is automatically set to "1" for a new channel.

Example:

```
doffset 100      # set the delay RAM address counter to 100
```

4.2.1.3.7 ELSE

See section *IF..ELSE*

4.2.1.3.8 EXT

Syntax: *time EXT channel level*

Syntax Graph: SIMULATE / STIMULUS

RAM area: none (simulator directive)

Description:

The simulator command *EXT* causes the simulation tool to set the input level of a given *channel* at a certain *time* to the specified *level*.

The channel must be in input mode, or an error will be reported by the simulator.

Time must be given in the format defined by the last *UNIT* command.

Level can be L (low), H (high) or X (undefined).

Example:

```
1000 ext 6 l     # at time 1000 UNITs set channel 6 input low
```

4.2.1.3.9 FALL

Syntax: *FALL*

Syntax Graph: CHANNEL / STATE / ACTION

RAM area: state RAM / CONDITION command word

Code:	D31..30	D29..15	D14	D13..11	D10	D9..7	D6	D5	D4..0
(IF)	-	-	0	001	-	-	-	-	-

Code:	D31..30	D29..15	D14	D13..11	D10	D9..7	D6	D5	D4..0
(ELSE)	-	-	-	-	0	001	-	-	-

Description:

The *FALL* command causes the actual channel to stay in or change to input mode. A delay definition in this state is not needed and will be ignored. The channel will stay in this condition until the input level at the according pin changes from high to low (falling edge enabled). Then the next state of this channel will be executed.

Example:

```
    fall, wait;      # stay in input mode until a falling edge is detected
```

4.2.1.3.10 FLAG

FLAG is a condition for the *IF* command. See section *IF .. ELSE*

4.2.1.3.11 GOTO

Syntax: *GOTO label*

Syntax Graph: CHANNEL / STATE / ACTION

RAM area: state RAM / NEXT command word

Code:	D31..27	D26..7	D6..0
(IF)	-	-	label value

Code:	D31..27	D26..22	D21..15	D14..0
(ELSE)	-	-	label value	-

Description:

GOTO causes the channel controller to continue execution at a state defined by a *label*.

*GOTO*s are only allowed to *labels* within the same channel.

If no *GOTO* is given, execution will proceed at the next state.

If a state contains two different actions (by using an *IF / ELSE* command), then
- *GOTO*s to different *labels* or the same *label* may be given, or
- one action may proceed with a *GOTO*, and the other continue, or
- both actions may continue with the next state.

Example:

```
label1:              # define label1
...
... goto label1;     # branch to label1
...
... IF FLAG1 .. goto label1 ELSE ... goto label2;
```

4.2.1.3.12 IF .. ELSE

Syntax:	*IF condition action1 ELSE action2;*
Syntax Graph:	CHANNEL / STATE
RAM area:	state RAM / CONDITION command word

Code:

D31	D30	D29..15	D14	D13..11	D10	D9..7	D6	D5	D4..0
-	COND5	-	-	-	-	-	-	-	COND4..0

Description:

The *IF .. ELSE* command allows conditional branches. Condition may be
- the input or output level at any channel pin (*IO*), if the corresponding module is inserted,
- one of 7 (from board version DS5101-04 on: 31) flags (*FLAG*) which can be set and reset by software via the PHS bus,
- or the *ZERO* flag of the loop counter (see *LDCTR* and *DECCTR*).

Condition is coded as

Code:	condition:	syntax:
0x00..0x0F:	IO Channels 1 .. 16	*IO* n
0x10..0x16:	PHS-Flags 1 .. 7	*FLAG* n
0x18:	loop counter zero	*ZERO*
0x1F:	always	(no *IF .. ELSE*)
0x20..0x37:	PHS-Flags 8 .. 31	*FLAG* n (only DS5101-04 and later)
0x38..0x3F:	always	-

The *IF .. ELSE* command must contain two complete actions. The first action is terminated by the *ELSE* command, the second by the end-of-state character ’;’. If the condition is true, the first action is executed, otherwise the second one.

Example:

```
    if io5 set t1,   # if channel 5 is high, then set output high for t1
     else reset t2;  # else set output low for t2
```

4.2.1.3.13 IMMEDIATE

Syntax: *IMMEDIATE*

Syntax Graph: CHANNEL

RAM area: state RAM / NEXT command word at state 0

Code:

D31..27	D26	D25..12	D11	D10..0
-	1	-	1	-

Description:

IMMEDIATE sets the channel to immediate update mode.

The command *IMMEDIATE* must be given as the first instruction within a channel definition if required. It is coded in the NEXT command word of state 0, which is executed immediately after channel reset. It is valid for all parameter updates in this channel.

The immediate update mode causes the channel controller to always use the most recently updated delay buffer. This allows to abort or increase a current time delay immediately, for example if a very low frequency is changed to a higher frequency in a D/F application.

If no *IMMEDIATE* is given, switching to an updated delay buffer is enabled by the *SYNCEN* command. If neither *IMMEDIATE* nor *SYNCEN* commands are used, then no update of delays is possible.

Example:

```
ch1              # start definition of channel 1
immediate        # select immediate update mode
```

4.2.1.3.14 INITFUNC

Syntax: *INITFUNC name [channel delayname] ;*

Syntax Graph: FUNCTIONS

RAM area: none (compiler directive)

Description:

INITFUNC controls the generation of a ready-to-use C initialization function, which is generated by the compiler into the C output file.

This function may be used to initialize any delays prior to starting the channel controller. It must not be used to update delays while the controller is running.

The name of the C function will be
ds5101_(source file name)_init_(name).

An init function may cover any delays of any channels without restrictions.

The init function writes the data to all three delay buffers. Therefore the channel must be in reset state while the function is executed.

Channel must be in the range 1..16 or G for global delays. In the latter case, global is interpreted as ’all channels used in this source file’. The reason for this is that the update function shall not interfere with channels which may be used by other applications.

Example:

The delay *t_pulse* in channel 2 shall be initialized. The source file name is *demo.src*.

In *demo.src* these lines have to be added:

```
initfunc vars
 2 t_pulse;
```

The compiler will generate this init function:

```
void ds5101_demo_init_var (long  base,
                           float T_PULSE_2)
{
  volatile long *ds5101_adr  = (long *) (PHS_BUS_BASE + base + 0x0c);
  volatile long *ds5101_mudr = (long *) (PHS_BUS_BASE + base + 0x00);

  *ds5101_adr  = 0x00024802;
  *ds5101_mudr = (long) (T_PULSE_2 * 40e6);
  *ds5101_adr  = 0x00024A02;
  *ds5101_mudr = (long) (T_PULSE_2 * 40e6);
  *ds5101_adr  = 0x00024C02;
  *ds5101_mudr = (long) (T_PULSE_2 * 40e6);
  *ds5101_adr  = 0;
}
```

If the delays are stored at consecutive addresses, then the auto increment feature of the address register is used.

All delays must be given as float values scaled to seconds.

4.2.1.3.15 IO

IO is a condition for the *IF* command. See section *IF* .. *ELSE*

4.2.1.3.16 Labels

Syntax: *label:*

Syntax Graph: CHANNEL / STATE

RAM area: none (compiler directive)

Description:

Labels are used by the *GOTO* command to perform absolute branches. All statements which end with a ":" without separators in between are interpreted as *labels*.

labels may have up to 20 characters. A *label* may be defined for each state.

Example:

```
label1:          # define label ’label1’
ENTRY:
L_3436:
```

4.2.1.3.17 LDCTR

Syntax: *LDCTR value*

Syntax Graph: CHANNEL / STATE / ACTION

RAM area: state RAM / INDEX command word

Code:	D31..15	D14	D13..9	D8..0
(IF)	-	1	value4..0	-

Code:	D31..30	D29	D28..24	D23..0
(ELSE)	-	1	value4..0	-

RAM area: state RAM / NEXT command word

Code:	D31..15	D14	D13..10	D9..7	D6..0
(IF)	-	1	-	value7..5	-

Code:	D31..30	D29	D28..25	D24..22	D21..0
(ELSE)	-	1	-	value7..5	-

Description:

LDCTR loads the 8 bit channel loop counter with the specified value. Legal values are 0..255.

The channel loop counter can be decremented with the *DECCTR* command. The zero condition of the channel loop counter can be tested with the *IF ZERO* command.

Within an *IF .. ELSE* construction, different channel loop counter commands may be given in the *IF* and the *ELSE* section.

Example:

```
            CH1                     # Channel 1: generate 5 pulses with
                                    # t_low/t_high pulse widths
            delay t_low 30          # define t_low
            delay t_high 10         # define t_high
            ldctr 5, set t_high;    # counter=5, start first pulse
entry:      decctr, reset t_low;    # counter--1, stop pulse
            if zero (...)           # next action after 5 pulses
               else set t_high,     # start pulses 2-5
                    goto entry;     # and loop
```

4.2.1.3.18 LOOP

Syntax: *LOOP*

Syntax Graph: CHANNEL / STATE / ACTION

RAM area: state RAM / NEXT command word

Code:	D31..27	D26..7	D6..0
(IF)	-	-	0

Code:	D31..27	D26..22	D21..15	D14..0
(ELSE)	-	-	0	-

Description:

LOOP causes the channel controller to continue the execution of states for the actual channel with the first state. This command is used for programming endless loops.

Example:

```
ch1                     # simple oscillator
delay t1 100            # define t1 = 100 UNITs
toggle,t1,loop;         # toggle every 100 UNITs
```

It is equivalent to

```
ch1
delay t1 100            # define t1 = 100 UNITs
start:
toggle,t1,goto start; # toggle every 100 UNITs
```

4.2.1.3.19 MSEC

MSEC is a unit for delay definitions. See section *UNITS*

4.2.1.3.20 NONE

Syntax: *NONE*

Syntax Graph: CHANNEL / STATE / ACTION

RAM area: state RAM / CONDITION command word

Code:	D31..30	D29..15	D14	D13..11	D10	D9..7	D6	D5	D4..0
(IF)	-	-	0	000	-	-	-	-	-

Code:	D31..30	D29..15	D14	D13..11	D10	D9..7	D6	D5	D4..0
(ELSE)	-	-	-	-	0	000	-	-	-

Description:

The *NONE* command causes the actual channel to stay in or change to input mode.

A delay definition by *delayname* in this state is not needed and will be ignored.

The channel will stay in this condition until the controller is reset by the DSP or triggered by another channel. *NONE* is the default command for all channels if no other command (*SET*, *RESET*..) is given.

Example:

```
    none wait;      # disable the channel
```

4.2.1.3.21 NOP

Syntax: *NOP*

Syntax Graph: CHANNEL / STATE / ACTION

RAM area: state RAM / CONDITION command word

Code:	D31..30	D29..15	D14	D13..11	D10	D9..7	D6	D5	D4..0
(IF)	-	-	-	111	-	-	-	-	-

Code:	D31..30	D29..15	D14	D13..11	D10	D9..7	D6	D5	D4..0
(ELSE)	-	-	-	-	-	111	-	-	-

Description:

The *NOP* command causes the actual channel to stay in or change to output mode and keep its current output state.

The *NOP* command may be useful if other channels shall be triggered, but the triggering channel should not change its output state.

Example:

```
    nop t1;          # do not change the output for t1
```

4.2.1.3.22 NSEC

NSEC is a unit for delay definitions. See section *UNITS*

4.2.1.3.23 OFFSET

Syntax: *OFFSET state*

Syntax Graph: CHANNEL / STATE

RAM area: none (compiler directive)

Description:

OFFSET causes the compiler to set its internal state counter for the state buffer to a new value.

States will be stored at address
(*state*) * 4 + 0x0600 (see section 6.2),
because every state occupies three words of data. Every fourth word in the state buffer remains unused. The state counter is incremented afterwards. If the state buffer location is already in use, an error will be generated.

State has to be in the range of 1 to 127.

The state counter is set to "0" for each new channel in the source file. The state at address 0 is executed immediately after reset and therefore serves as an initialization state. An attempt to use the *OFFSET* command to skip this initialization state will result in an error message.

Example:

```
offset 100       # set the state RAM address counter to 100
```

4.2.1.3.24 PHSINT

Syntax: *PHSINT*

Syntax Graph: CHANNEL / STATE

RAM area: state RAM / CONDITION command word

Code:

D31..30	D29..15	D14	D13..11	D10	D9..7	D6	D5	D4..0
-	-	-	-	-	-	-	1	-

Description:

PHSINT sets the interrupt flag of the actual channel in the Interrupt Register (IR). If the corresponding Interrupt Mask Register (IMR) bit is set, then a PHS bus interrupt is generated.

Example:

```
phsint, set t1;  # generate a DSP interrupt, set output high for t1
```

4.2.1.3.25 READFUNC

Syntax: *READFUNC name [channel delayname] ;*

Syntax Graph: FUNCTIONS

RAM area: none (compiler directive)

Description:

READFUNC controls the generation of a ready-to-use C read function, which is included in the compiler C output file.

The name of the C function will be
ds5101_(source file name)_read_(name).

A read function may read any delays of any channels without restrictions. The swinging buffer controller ensures that the delays are read from the most recently updated buffer.

Channel must be in the range 1..16 or G for global delays. In the latter case, parameters are read from the lowest channel used in this source file.

Example:

The delays *t_main* in channel 1 and *t_pulse* and *t_1* in channel 2 shall be read back. The source file name is *demo.src*.

In *demo.src* these lines have to be added:

```
readfunc vars
 1 t_main
 2 t_pulse
 2 t_1 ;
```

The compiler will generate this read function:

```
void ds5101_demo_read_var (long  base,
                          *float T_MAIN_1,
                          *float T_PULSE_2,
                          *float T_1_2)
{
  volatile long *ds5101_adr  = (long *) (PHS_BUS_BASE + base + 0x0c);
  volatile long *ds5101_mudr = (long *) (PHS_BUS_BASE + base + 0x00);

  *ds5101_adr        = 0x00014201;
  *T_MAIN_1          = (float) ( *ds5101_mudr ) * 25e-9 ;
  *ds5101_adr        = 0x00024202;
  *T_PULSE_2         = (float) ( *ds5101_mudr ) * 25e-9 ;
  *T_1_2             = (float) ( *ds5101_mudr ) * 25e-9 ;
  *ds5101_adr        = 0;
}
```

If the delays are stored at consecutive addresses, then the auto increment feature of the address register is used (cf. t_pulse and t_1 in the example above).

All delays will be returned as float values scaled to seconds.

4.2.1.3.26 RESET

Syntax: *RESET*

Syntax Graph: CHANNEL / STATE / ACTION

RAM area: state RAM / CONDITION command word

Code:	D31..30	D29..15	D14	D13..11	D10	D9..7	D6	D5	D4..0
(IF)	-	-	-	101	-	-	-	-	-

Code:	D31..30	D29..15	D14	D13..11	D10	D9..7	D6	D5	D4..0
(ELSE)	-	-	-	-	-	101	-	-	-

Description:

The *RESET* command causes the actual channel to stay in or change to output mode and set its output level low.

Example:

```
    reset t1;        # set output low for t1
```

4.2.1.3.27 RISE

Syntax: *RISE*

Syntax Graph: CHANNEL / STATE / ACTION

RAM area: state RAM / CONDITION command word

Code:	D31..30	D29..15	D14	D13..11	D10	D9..7	D6	D5	D4..0
(IF)	-	-	0	010	-	-	-	-	-

Code:	D31..30	D29..15	D14	D13..11	D10	D9..7	D6	D5	D4..0
(ELSE)	-	-	-	-	0	010	-	-	-

Description:

The *RISE* command causes the actual channel to stay in or change to input mode. A delay definition by *delayname* is not needed and will be ignored. The channel will stay in this condition until the input level at the according pin changes from low to high. Then the next state will be executed.

Example:

```
    rise, wait;      # stay in input mode until a rising edge is detected
```

4.2.1.3.28 SEC

SEC is a unit for delay definitions. See section *UNITS*

4.2.1.3.29 SET

Syntax: *SET*

Syntax Graph: CHANNEL / STATE / ACTION

RAM area: state RAM / CONDITION command word

Code:	D31..30	D29..15	D14	D13..11	D10	D9..7	D6	D5	D4..0
(IF)	-	-	-	110	-	-	-	-	-

Code:	D31..30	D29..15	D14	D13..11	D10	D9..7	D6	D5	D4..0
(ELSE)	-	-	-	-	-	110	-	-	-

Description:

The *SET* command causes the actual channel to stay in or change to output mode and set its output level high.

Example:

```
    set t1;          # set output high for t1
```

4.2.1.3.30 SIMULATE

Syntax:	*SIMULATE*
Syntax Graph:	(root)
RAM area:	none (compiler directive)

Description:

SIMULATE finishes the channel definitions and starts the simulator directives section.

Legal commands within this section are:

WR	(write delay RAM)
EXT	(set the external input level)
UPDATE	(set the UPDATE-ENABLE bit in the flag register FLR)
SWRES	(software reset)
SWTR	(software trigger)

Example:

```
simulate        # enter simulator directives section
```

4.2.1.3.31 SWFL

Syntax: *time SWFL flag level*

Syntax Graph: SIMULATE / STIMULUS

RAM area: none (simulator directive)

Description:

The simulator command *SWFL* causes the simulation tool to set a *flag* bit in the flag register FLR at a certain *time* to the specified *level*.

Flags can be used by the *IF .. ELSE* command to perform conditional branches.

Time must be given in the format defined by the last *UNIT* command.

Level can be L (low) or H (high).

Example:

```
1000 swfl 6 h      # at time 1000 UNITs set software flag 6 high
```

4.2.1.3.32 SWRES

Syntax: *time SWRES channel level*

Syntax Graph: SIMULATE / STIMULUS

RAM area: none (simulator directive)

Description:

The simulator command *SWRES* causes the simulation tool to set a bit in the reset register RR at a certain *time* to the specified *level*.

If set to high level, the *channel* will stop operation immediately and act like an input channel.

The reset register RR is preset on power up, so that all channels are disabled.

Time must be given in the format defined by the last *UNIT* command.

Level can be L (low) or H (high).

Example:

```
1000 swres 6 h    # at time 1000 UNITs disable channel 6
2000 swres 8 l    # at time 2000 UNITs start channel 8
```

4.2.1.3.33 SWTR

Syntax:	*time SWTR channel*
Syntax Graph:	SIMULATE / STIMULUS
RAM area:	none (simulator directive)

Description:

The simulator command *SWTR* causes the simulation tool to set a bit in the software trigger register STR for a given *channel* at a certain *time*, thus generating a trigger event.

The channel must be either in input mode (*RISE*, *FALL*, *BOTH* or *NONE*) or in *WAIT* mode.

Time must be given in the format defined by the last *UNIT* command.

Example:

```
1000 swtr 6        # at time 1000 UNITs trigger channel 6
```

4.2.1.3.34 SYNCEN

Syntax: *SYNCEN*

Syntax Graph: CHANNEL / STATE

RAM area: state RAM / CONDITION command word

Code:

D31..30	D29..15	D14	D13..11	D10	D9..7	D6	D5	D4..0
-	-	-	-	-	-	1	-	-

Description:

SYNCEN causes the swinging buffer controller to proceed to the next delay buffer, if new data is available.

This command should be used at the beginning of a new output timing cycle to ensure that new data, which has been updated by the PHS bus master during the execution of a program, will be used from this time on.

Example:

```
    syncen, set t1;  # use new delay data, set output high for t1
```

4.2.1.3.35 TICKS

TICKS is a unit for delay definitions. See section *UNITS*

4.2.1.3.36 TOGGLE

Syntax:	*TOGGLE*
Syntax Graph:	CHANNEL / STATE / ACTION
RAM area:	state RAM / CONDITION command word

Code:	D31..30	D29..15	D14	D13..11	D10	D9..7	D6	D5	D4..0
(IF)	-	-	-	100	-	-	-	-	-

Code:	D31..30	D29..15	D14	D13..11	D10	D9..7	D6	D5	D4..0
(ELSE)	-	-	-	-	-	100	-	-	-

Description:

The *TOGGLE* command causes the actual channel to stay in or change to output mode and change its output level from high to low or from low to high depending on the current state.

Example:

```
    toggle t1;       # change output level for t1
```

4.2.1.3.37 TR (Trigger)

Syntax: *TRn*
TR n

Syntax Graph: CHANNEL / STATE

RAM area: state RAM / CONDITION command word

Code:

D31..30	D29..15	D14	D13..11	D10	D9..7	D6	D5	D4..0
-	see text	-	-	-	-	-	-	-

Description:

TR defines which other channels shall be triggered, i.e. shall proceed to their next state, regardless whether their delay has already passed.

These other channels should be in input mode or in the *WAIT* condition at this time.

A separator between *CH* and *n* may be inserted.

n must be in the range from 1 to 16, except for the actual channel number, because a channel cannot trigger itself.

Up to 15 *TR* commands may be given in one state. Every *TR* command is coded as a "1" in the CONDITION command word.

The actual coding of *TR* commands into CONDITION command word bits D29..15 depends on the actual channel number and is explained in detail in section 6.2.1.4.

The triggered channel needs some time to process the trigger signal. There is a constant delay of 6 *TICKs* (150ns) before a channel reacts on a trigger event. In order to compensate this delay, each delay constant used in the triggered channel after a wait command is automatically reduced by 6 *TICKs* by the controller.

Example:

```
UNIT TICKS
DELAY T1 20
CH1  ....  TR2, SET ....
CH2  ....  WAIT; SET T1; RESET ...
```

will not generate a 20 *TICK* active high pulse on channel 2, but only a 14 *TICK* pulse. It starts 6 *TICKs* after the edge on channel 1, and ends 20 *TICKS* after the edge on channel 1.
If a pulse width of exactly *(n) TICKs* is needed after the trigger, and a delay of 16 *TICKs* from trigger is acceptable, then write

```
UNIT TICKS
DELAY T0 16
DELAY T1 (n)
CH1  ....  TR2, TOGGLE, ....
CH2  ....  WAIT; NOP T0; SET T1; RESET ...
```

which generates an *(n) TICK* pulse with a 16 *TICK* delay from channel 1. The 16 *TICKs* result from the minimum delay time of 10 *TICKs* + 6 *TICKs* trigger reaction time.

Example:

```
    tr6,tr8,set t1;  # trigger channels 6 and 8, set output high for t1
```

4.2.1.3.38 UNIT

Syntax:	*UNIT base*
Syntax Graph:	DELAYS CHANNEL / STATE / DELAYS
RAM area:	none (compiler directive)

Description:

UNIT defines the basic time unit for all delays and simulation commands.

Base may be one of this list:

TICKS	base unit = 1 machine cycle = 25ns
NSEC	base unit = 1 nanosecond = 1/25 TICK
USEC	base unit = 1 microsecond = 40 TICKs
MSEC	base unit = 1 millisecond = 40 000 TICKs
SEC	base unit = 1 second = 40 000 000 TICKs

UNIT may be redefined any time and is valid until the next redefinition. However, a warning is issued.

Example:

```
unit usec        # select microseconds as basic time unit
delay t1 20      # define t1 = 20µs = 800 TICKs
```

4.2.1.3.39 UPDATE

Syntax: *time UPDATE channel*

Syntax Graph: SIMULATE / STIMULUS

RAM area: none (simulator directive)

Description:

The simulator command *UPDATE* causes the simulation tool to set the update enable input of a given *channel* to high level at a certain *time*. Setting to low level is done by using the *WR* command.

If set to high level, the swinging buffer controller of the selected channel will proceed to the next delay buffer. In *IMMEDIATE* mode, the new data become effective immediately. Current pulses may be aborted or extended to the new value.
In *SYNCEN* mode, new data will become effective only when the next *SYNCEN* command is encountered.

For more information, see section 6.1.8.

Time must be given in the format defined by the last *UNIT* command.

Example:

```
    1000 update 6    # at time 1000 UNITs set UPDATE bit for channel 6
```

4.2.1.3.40 UPDFUNC

Syntax: *UPDFUNC name [channel delayname] ;*

Syntax Graph: FUNCTIONS

RAM area: none (compiler directive)

Description:

UPDFUNC controls the generation of a ready-to-use C update function by the compiler, which is included in the compiler C output file.

This function may be used to update delay values while the controller is running.

The name of the C function will be
ds5101_(source file name)_update_(name).

To ensure data consistency in the swinging buffer update system, all delay parameters of a certain channel must be updated by using a single update function.

Channel must be in the range 1..16 or G for global delays. In the latter case, global is interpreted as ’all channels used in this source file’. The reason for this is that the update function shall not interfere with channels which may be used by other applications.

Example:

The delays *t_main* in channel 1 and *t_pulse* and *t_1* in channel 2 shall be updated. The source file name is *demo.src*.

In *demo.src* these lines have to be added:

```
updfunc vars
 1 t_main
 2 t_pulse
 2 t_1 ;
```

The compiler will generate this update function:

```
void ds5101_demo_update_vars (long  base,
                              float T_MAIN_1,
                              float T_PULSE_2,
                              float T_1_2)
{
  volatile long *ds5101_adr  = (long *) (PHS_BUS_BASE + base + 0x0c);
  volatile long *ds5101_mudr = (long *) (PHS_BUS_BASE + base + 0x00);
  volatile long *ds5101_flr  = (long *) (PHS_BUS_BASE + base + 0x04);

  *ds5101_adr  = 0x00014001;
  *ds5101_mudr = (long) (T_MAIN_1 * 40e6);
  *ds5101_adr  = 0x00024002;
  *ds5101_mudr = (long) (T_PULSE_2 * 40e6);
  *ds5101_mudr = (long) (T_1_2 * 40e6);
  *ds5101_adr  = 0x00030000;        /* set channel select bits ch1+2 */
  *ds5101_flr |= 0x80;              /* set update enable bit         */
  *ds5101_adr  = 0;
}
```

If the delays are stored at consecutive addresses, then the auto increment feature of the address register is used (cf. t_pulse and t_1 in the example above).

All delays must be given as float values scaled to seconds.

Please note that the UPDATE bit in the flag register FLR is set at the end of the function with both channel select bits for channels 1 and 2 in the address register set. This ensures consistent data sets for all involved channels. For a detailed description please refer to section 6.1.8. ’*Address Register (AR)*’.

4.2.1.3.41 USEC

USEC is a unit for delay definitions. See section *UNITS*

4.2.1.3.42 WAIT

Syntax: *WAIT*

Syntax Graph: CHANNEL / STATE / ACTION

RAM area: state RAM / CONDITION command word

Code:	D31..30	D29..15	D14	D13..11	D10	D9..7	D6	D5	D4..0
(IF)	-	-	1	-	-	-	-	-	-

Code:	D31..30	D29..15	D14	D13..11	D10	D9..7	D6	D5	D4..0
(ELSE)	-	-	-	-	1	-	-	-	-

Description:

A *WAIT* command causes the controller to wait for a trigger event before proceeding to the next state. A trigger may be issued by the PHS bus master or by another channel.

When leaving *WAIT* due to a trigger event, the first delay constant used is reduced by 6 TICKs to compensate the trigger reaction time. See section 5.2.1.3.27 (*TRIGGER*) for further information.

Example:

```
    set wait;        # set output high and wait for trigger
```

4.2.1.3.43 WR

Syntax:	*time WR channel delayname value*
Syntax Graph:	SIMULATE / STIMULUS
RAM area:	none (simulator directive)

Description:

The simulator command *WR* causes the simulation tool to write a new *value* to the delay buffer RAM.

The swinging buffer controller decides, in which buffer (A, B or C) the data will be written.

Time and *value* must be given in the format defined by the last UNIT command.

Delayname must have been defined for the selected channel.

Channel numbers may be 1..16, or G. In the latter case, the new value is written to all 16 channels.

Examples:	1000 WR 6 t_1 100	(at time 1000 units write delay 100 units to delay t_1 at channel 6)
	5000 wr g t_g 200	(at time 5000 units write delay 200 units to delay t_g at all 16 channels

4.2.2 File Formats

4.2.2.1 Source File

The source file must be a plain ascii text file. The syntax is described in the previous chapters of this documentation.

It must have the extension *.src*.

The number of characters per line is limited to 255.

Example:

```
#-----------------------------------------
# Source file example for DS5101 DWO
# 12.12.95 By
# 1 phase pwm
#-----------------------------------------

UNIT TICKS               # all delays are given in 25ns ticks
delay t0 10              # minimum pulse width

CH 1                     # channel 1: master clock running at 100kHz

delay t_master 200       # half period = 1/100kHz / 25nsec / 2 = 200 Ticks

  reset t0;              # low after reset

begin:
  syncen,                # enable use of new data from DSP
    phsint,              #  generate interrupt for DSP
    tr2,tr3,             #  trigger channels 2 and 3
    toggle, t_master;    #  toggle output, wait half period

  toggle, t_master,      # toggle output, wait half period,
    goto begin;          #  loop


CH2                      # channel 2: phase A - highside

delay t_low  200
delay t_high 200

  reset,                 # low after reset
  wait;                  # state 1: wait for trigger
begin:
  syncen, nop t_low;    # state 2: wait t_low with output low
  set t_high;           # state 3: set output high and wait for t_high
  reset wait goto begin;# state 4: set output low and jump to begin

CH3                     # channel 3: phase A - lowside

delay t_high 195
delay t_low  205

  set,                  # high after reset
  wait;                 # state 1: wait for trigger
begin:
  syncen, nop t_high;   # state 2: wait t_high with output high
  reset t_low;          # state 3: set output low and wait for t_low
  set wait goto begin;  # state 4: set output high and jump to begin

initfunc cycle
 1 t_master;

updfunc pwm
 2 t_low, 2 t_high,
 3 t_high, 3 t_low;

readfunc all
 1 t_master,
 2 t_low, 2 t_high,
 3 t_high, 3 t_low;

simulate

1000 wr 2 t_low  100   # change duty cycle to 75%
1010 wr 2 t_high 300
1020 wr 3 t_high  95
1030 wr 3 t_low  305
1040 update 2
1040 update 3
```

4.2.2.2 List File

The list file is a ascii text file containing all lines of the source file plus all messages from the compiler.

The extension for the list file is *.lst*.

Every time the compiler has detected a completed state definition, a report line is generated showing the binary code which the compiler has generated.
Example:

```
CH1  STATE: 0001  COND: 0003A04F  INDEX: 00000002  NEXT: 00000002
```

All values (except the channel number) are given in hexadecimal notation.

CH:	Channel number = 1 (1..16)
STATE:	State counter = 0x0001
COND:	Condition command word = 0x0003a04f
INDEX:	Index command word = 0x00000002
NEXT:	Next command word = 0x00000002

If a new delay constant has been defined, an additional report line is generated:

```
CH2     DELAY 0002: 0000005C (TA1)
```

which means:
Delay at delay buffer address 2 of channel 2 = 0x0000005c = 92 Ticks = 2.3µsec

Example (from the source file in the previous section):

```
DWO Compiler V2.0      12.11.96

Source  = TEST.SRC
Listing = TEST.LST
Result  = TEST.DAT


--- PASS 1 ---

GLOBAL  DELAY 0001: 0000000A (T0)
CH1     DELAY 0002: 000000C8 (T_MASTER)
CH2     DELAY 0002: 000000C8 (T_LOW)
CH2     DELAY 0003: 000000C8 (T_HIGH)
CH3     DELAY 0002: 000000C3 (T_HIGH)
CH3     DELAY 0003: 000000CD (T_LOW)

--- PASS 2 ---
```

```
#---------------------------------------
# Source file example for DS5101 DWO
# 12.12.95 By
# 1 phase pwm
#---------------------------------------

UNIT TICKS              # all delays are given in 25ns ticks
delay t0 10             # minimum pulse width

CH 1                    # channel 1: master clock running at 100kHz

delay t_master 200      # half period = 1/100kHz / 25nsec / 2 = 200 Ticks

  reset t0;
CH1  STATE: 0000  COND: 00002A9F  INDEX: 00008001  NEXT: 00008001
             # low after reset

begin:
  syncen,               # enable use of new data from DSP
    phsint,             #  generate interrupt for DSP
    tr2,tr3,            #  trigger channels 2 and 3
    toggle, t_master;
CH1  STATE: 0001  COND: 0001A27F  INDEX: 00010002  NEXT: 00010002
   #  toggle output, wait half period

  toggle, t_master,     # toggle output, wait half period,
    goto begin;
CH1  STATE: 0002  COND: 0000221F  INDEX: 00010002  NEXT: 00008001
         #  loop


CH2                     # channel 2: phase A - highside

delay t_low  200
delay t_high 200

  reset,                # low after reset
  wait;
CH2  STATE: 0000  COND: 00006E9F  INDEX: 00000000  NEXT: 00008001
                 # state 1: wait for trigger
begin:
  syncen, nop t_low;
CH2  STATE: 0001  COND: 00003BBF  INDEX: 00010002  NEXT: 00010002
    # state 2: wait t_low with output low
  set t_high;
CH2  STATE: 0002  COND: 0000331F  INDEX: 00018003  NEXT: 00018003
           # state 3: set output high and wait for t_high
  reset wait goto begin;
CH2  STATE: 0003  COND: 00006E9F  INDEX: 00000000  NEXT: 00008001
# state 4: set output low and jump to begin

CH3                     # channel 3: phase A - lowside
```

```
delay t_high 195
delay t_low  205

  set,                  # high after reset
  wait;
CH3  STATE: 0000  COND: 0000771F  INDEX: 00000000  NEXT: 00008001
                 # state 1: wait for trigger
begin:
  syncen, nop t_high;
CH3  STATE: 0001  COND: 00003BBF  INDEX: 00010002  NEXT: 00010002
   # state 2: wait t_high with output high
  reset t_low;
CH3  STATE: 0002  COND: 00002A9F  INDEX: 00018003  NEXT: 00018003
          # state 3: set output low and wait for t_low
  set wait goto begin;
CH3  STATE: 0003  COND: 0000771F  INDEX: 00000000  NEXT: 00008001
  # state 4: set output high and jump to begin

initfunc cycle
 1 t_master;

updfunc pwm
 2 t_low, 2 t_high,
 3 t_high, 3 t_low;

readfunc all
 1 t_master,
 2 t_low, 2 t_high,
 3 t_high, 3 t_low;

simulate


1000 wr 2 t_low  100   # change duty cycle to 75%
1010 wr 2 t_high 300
1020 wr 3 t_high  95
1030 wr 3 t_low  305
1040 update 2
1040 update 3


Symbol table

Name                  Channel  Address
BEGIN                    1     0001
BEGIN                    2     0001
BEGIN                    3     0001

Errors:   0
Warnings: 0
execution completed successfully
```

4.2.2.3 Simulator Data File

The compiler generates an output file for the simulator tool DWOSIM. It has the extension *.dat*.

This file uses a binary file format and therefore can not be edited with a text editor.

It contains the same data as the init function in the C source code output file, plus additional information from the *SIMULATE* section of the source.

4.2.2.4 C Source Code Output File

The compiler generates a C source code file which can be included in C applications. This file has the extension *.c*.

In addition, a header file with the extension *.h* is generated, which contains defines for all delays used in the source. These defines can be used to access delay values without any knowledge about the actual RAM addresses.

Note: For board version DS5101-04 and later, a check routine is inserted when using DWOCOMP V3.0.2. If any FLAG between 8 and 31 is used, and the boards version is earlier than DS5101-04, then an error message is generated.

Example:

```
/************************************************************************
       C source file for DS5101 download from TEST.SRC
************************************************************************/

#include "dsperror.h"

void ds5101_test_init_cycle (long  base,
                             float T_MASTER_1)
{
  volatile long *ds5101_adr  = (long *) (PHS_BUS_BASE + base + 0x0c);
  volatile long *ds5101_mudr = (long *) (PHS_BUS_BASE + base + 0x00);
  volatile long *ds5101_flr  = (long *) (PHS_BUS_BASE + base + 0x04);
  *ds5101_adr  = 0x00012802;
  *ds5101_mudr = (long) (T_MASTER_1 * 40e6);
  *ds5101_adr  = 0x00012A02;
  *ds5101_mudr = (long) (T_MASTER_1 * 40e6);
  *ds5101_adr  = 0x00012C02;
  *ds5101_mudr = (long) (T_MASTER_1 * 40e6);
}

void ds5101_test_update_pwm (long  base,
                             float T_LOW_2,
                             float T_HIGH_2,
                             float T_HIGH_3,
                             float T_LOW_3)
{
  volatile long *ds5101_adr  = (long *) (PHS_BUS_BASE + base + 0x0c);
  volatile long *ds5101_mudr = (long *) (PHS_BUS_BASE + base + 0x00);
  volatile long *ds5101_flr  = (long *) (PHS_BUS_BASE + base + 0x04);

  *ds5101_adr  = 0x00022002;
  *ds5101_mudr = (long) (T_LOW_2 * 40e6);
  *ds5101_mudr = (long) (T_HIGH_2 * 40e6);
  *ds5101_adr  = 0x00042002;
  *ds5101_mudr = (long) (T_HIGH_3 * 40e6);
  *ds5101_mudr = (long) (T_LOW_3 * 40e6);
  *ds5101_adr  = 0x00060000;
  *ds5101_flr |= 0x80;
}

void ds5101_test_read_all (long  base,
                           float *T_MASTER_1,
                           float *T_LOW_2,
                           float *T_HIGH_2,
                           float *T_HIGH_3,
                           float *T_LOW_3)
{
  volatile long *ds5101_adr  = (long *) (PHS_BUS_BASE + base + 0x0c);
  volatile long *ds5101_mudr = (long *) (PHS_BUS_BASE + base + 0x00);
  volatile long *ds5101_flr  = (long *) (PHS_BUS_BASE + base + 0x04);

  *ds5101_adr        = 0x00012202;
  *T_MASTER_1        = (float) ( *ds5101_mudr ) * 25e-9 ;
  *ds5101_adr        = 0x00022202;
  *T_LOW_2           = (float) ( *ds5101_mudr ) * 25e-9 ;
  *T_HIGH_2          = (float) ( *ds5101_mudr ) * 25e-9 ;
  *ds5101_adr        = 0x00042202;
  *T_HIGH_3          = (float) ( *ds5101_mudr ) * 25e-9 ;
  *T_LOW_3           = (float) ( *ds5101_mudr ) * 25e-9 ;
}
```

```
int ds5101_test_load (long base, long mask)
{
  volatile long *ds5101_rr   = (long *) (PHS_BUS_BASE + base + 0x09);
  volatile long *ds5101_adr  = (long *) (PHS_BUS_BASE + base + 0x0c);
  volatile long *ds5101_mudr = (long *) (PHS_BUS_BASE + base + 0x00);
  volatile long *ds5101_flr  = (long *) (PHS_BUS_BASE + base + 0x04);
  volatile long *ds5101_stp  = (long *) (PHS_BUS_BASE + base + 0x05);
  {
    static int ptr, count, i, error;
    static unsigned long data_ds5101[] = {
      0x00012E00,     /* state address channel 1 */
      0x0000000C,     /* count */
      0x00002A9F,0x00008001,0x00008001,0x00000000,
      0x0001A27F,0x00010002,0x00010002,0x00000000,
      0x0000221F,0x00010002,0x00008001,0x00000000,
      0x00022E00,     /* state address channel 2 */
      0x00000010,     /* count */
      0x00006E9F,0x00000000,0x00008001,0x00000000,
      0x00003BBF,0x00010002,0x00010002,0x00000000,
      0x0000331F,0x00018003,0x00018003,0x00000000,
      0x00006E9F,0x00000000,0x00008001,0x00000000,
      0x00042E00,     /* state address channel 3 */
      0x00000010,     /* count */
      0x0000771F,0x00000000,0x00008001,0x00000000,
      0x00003BBF,0x00010002,0x00010002,0x00000000,
      0x00002A9F,0x00018003,0x00018003,0x00000000,
      0x0000771F,0x00000000,0x00008001,0x00000000,
      0x00012800,     /* delay address buffer A channel 1 */
      0x00000002,     /* count */
      0x00000000, 0x0000000A,
      0x00022800,     /* delay address buffer A channel 2 */
      0x00000002,     /* count */
      0x00000000, 0x0000000A,
      0x00042800,     /* delay address buffer A channel 3 */
      0x00000002,     /* count */
      0x00000000, 0x0000000A,
      0x00012A00,     /* delay address buffer B channel 1 */
      0x00000002,     /* count */
      0x00000000, 0x0000000A,
      0x00022A00,     /* delay address buffer B channel 2 */
      0x00000002,     /* count */
      0x00000000, 0x0000000A,
      0x00042A00,     /* delay address buffer B channel 3 */
      0x00000002,     /* count */
      0x00000000, 0x0000000A,
      0x00012C00,     /* delay address buffer C channel 1 */
      0x00000002,     /* count */
      0x00000000, 0x0000000A,
      0x00022C00,     /* delay address buffer C channel 2 */
      0x00000002,     /* count */
    0x00000000, 0x0000000A,
    0x00042C00,     /* delay address buffer C channel 3 */
    0x00000002,     /* count */
    0x00000000, 0x0000000A,
    0x00012802,     /* delay address buffer A channel 1 */
    0x00000001,     /* count */
    0x000000C8,
    0x00012A02,     /* delay address buffer B channel 1 */
    0x00000001,     /* count */
    0x000000C8,
    0x00012C02,     /* delay address buffer C channel 1 */
    0x00000001,     /* count */
    0x000000C8,
    0x00022802,     /* delay address buffer A channel 2 */
    0x00000002,     /* count */
    0x000000C8, 0x000000C8,
    0x00022A02,     /* delay address buffer B channel 2 */
    0x00000002,     /* count */
    0x000000C8, 0x000000C8,
    0x00022C02,     /* delay address buffer C channel 2 */
    0x00000002,     /* count */
    0x000000C8, 0x000000C8,
    0x00042802,     /* delay address buffer A channel 3 */
    0x00000002,     /* count */
    0x000000C3, 0x000000CD,
    0x00042A02,     /* delay address buffer B channel 3 */
    0x00000002,     /* count */
    0x000000C3, 0x000000CD,
    0x00042C02,     /* delay address buffer C channel 3 */
    0x00000002,     /* count */
    0x000000C3, 0x000000CD,
    0x00000000 };   /* end of data */

  /* download routine */

  *ds5101_rr |= mask;

  ptr = 0;
  while (data_ds5101[ptr])
  {
    *ds5101_adr = data_ds5101[ptr++];
    count = data_ds5101[ptr++];
    if (*ds5101_adr & (mask << 16))
      for (i=0; i<count; i++)
        *ds5101_mudr = data_ds5101[ptr++];
    else
      ptr += count;
  }
  ptr   = 0;
  error = 0;
  while (data_ds5101[ptr])
```

```
    {
      *ds5101_adr = data_ds5101[ptr++];
      count = data_ds5101[ptr++];
      if (*ds5101_adr & (mask << 16))
      {
        for (i=0; i<count; i++)
          if (*ds5101_mudr != data_ds5101[ptr++])
          {
            error = *ds5101_adr & 0xffff0000;
            if (error == 0x80000000) return DS5101_16_LOAD_ERROR;
            if (error == 0x40000000) return DS5101_15_LOAD_ERROR;
            if (error == 0x20000000) return DS5101_14_LOAD_ERROR;
            if (error == 0x10000000) return DS5101_13_LOAD_ERROR;
            if (error == 0x08000000) return DS5101_12_LOAD_ERROR;
            if (error == 0x04000000) return DS5101_11_LOAD_ERROR;
            if (error == 0x02000000) return DS5101_10_LOAD_ERROR;
            if (error == 0x01000000) return DS5101_9_LOAD_ERROR;
            if (error == 0x00800000) return DS5101_8_LOAD_ERROR;
            if (error == 0x00400000) return DS5101_7_LOAD_ERROR;
            if (error == 0x00200000) return DS5101_6_LOAD_ERROR;
            if (error == 0x00100000) return DS5101_5_LOAD_ERROR;
            if (error == 0x00080000) return DS5101_4_LOAD_ERROR;
            if (error == 0x00040000) return DS5101_3_LOAD_ERROR;
            if (error == 0x00020000) return DS5101_2_LOAD_ERROR;
            return DS5101_1_LOAD_ERROR;
          }
      }
      else
        ptr += count;
    }
  }
  return NO_ERROR;
}
```

4.2.2.5 Error File

If the option "-e+" has been selected, the compiler will generate an error file *dwocomp.err* in case an error has occurred. The error file can contain several error messages.

If more than one errors are reported, concentrate on the first error. Further errors may follow the first one, if the compiler fails to proceed with the next statement.

Example:

```
***** ERROR 3019 in line 46: action: pin mode missing
Operand     : WAIT
***** ERROR 3008 in line 47: end of state (;) expected
Operand     : NOP
***** ERROR 3019 in line 47: action: pin mode missing
Operand     : TA4
```

4.2.3 Error Messages

1000: source file not found

DWOCOMP cannot find the source file specified as parameter %1. The file name must be given without extension (extension is always *.src*).
Correct syntax:

```
DWOCOMP filename [options]
```

1001: insufficient RAM

DWOCOMP needs about 2.3 MBytes of RAM for operation, running in protected mode. Make sure that the operating system (DOS$^{(r)}$ or WINDOWS$^{(r)}$) has enough RAM available.

1002: too many delays

A PC RAM overflow has occurred, because too many delays have been defined. Every delay needs about 30 additional bytes of heap RAM.

1003: too many delays

A delay RAM overflow has occurred. Up to 511 delays may be used.

1004: too many states

A state RAM overflow has occurred. Up to 128 states may be used per channel.

2000: label too long

Labels may have up to 20 characters.

2001: label not found

A *GOTO* command uses a *label* which has not been defined.

2002: illegal label

A separator (blank or tab) has been added between *label* and ":". Please remove.

2003: duplicate label

A *label* name has already been defined. Use a new name.

2004: duplicate label

Do not assign two different *labels* to the same state address.

3000: illegal channel

Channel must be in the range 1 to 16.

3001: channel already defined

Every channel may only be defined once. All states belonging to one channel must be defined in one sequence.

3002: illegal OFFSET

OFFSET must be in the range 1 to 127.

3003: first state must be at address 0

Do not use the *OFFSET* command to relocate the first state of a channel. This state must be stored at address 0, and it is executed immediately after reset.

3004: last state leads to undefined state

In the last state either in the *IF* or in the *ELSE* section (or both) no *GOTO* or *LOOP* command has been given, so the channel controller tries to continue at the next state. Do not move the RAM location of this next state with the *OFFSET* command. If you have to do so, use the *GOTO label* command in the preceeding state and add a *label* after the *OFFSET* command.

3005: state RAM address already used

OFFSET was used to set the state address counter to a state which has been already used.

3006: state RAM address already used

The state address which is needed for storing the next state is already occupied, propably by an earlier *OFFSET* command.

3007: ELSE definition missing

Did you terminate the *IF* part of an *IF .. ELSE* command with a ’;’?

3008: end of state (;) expected

The compiler expected an end of state character. See the syntax graph to find out what went wrong.

3009: illegal trigger

Trigger channel must be in the range 1 to 16.

3010: trigger on same channel not allowed

A channel may not trigger itself (of what use would this be?). Therefore an error is presumed in this case.

3011: illegal IF condition

Only use *CH n* (n=1..16), *FLAG n* (n=1..7) or *ZERO* as condition of an *IF* command.

3012: illegal IF FLAG condition

An illegal *FLAG* number was used as condition of an *IF .. ELSE* command. Only *FLAG* numbers 1..7 are legal.

3013: illegal IF I/O condition

An illegal channel number was used as condition of an *IF .. ELSE* command. Only channels 1..16 are legal.

3014: illegal delay

Delay is always scaled with the time base *UNIT*. The resulting value must be in the range 0 to 26.84 seconds. *UNIT* is set to *TICKS* (25 ns steps) by default, if undefined.

3015: illegal delay (format error)

The delay value could not be interpreted. Use real or integer numbers only.

3016: illegal DELAY OFFSET

DOFFSET must be in the range 1 to 511.

3017: DELAY RAM address already used

DOFFSET was used to set the delay address counter to an address which is already used.

3018: illegal UNIT

UNIT must be one of *TICKS*, *NSEC*, *USEC*, *MSEC* or *SEC*

3019: action pin mode missing

Every action must contain one of *SET*, *RESET*, *TOGGLE*, *NOP*, *RISE*, *FALL*, *BOTH* or *NONE*.

3020: last state leads to undefined state

Use *LOOP* or *GOTO* with the last state within a channel to ensure proper program flow.

3021: delay not found

The delay used within the action has not been defined.

3022: delayname already used as global delay

The *delayname* has already been used as a global delay. Choose another name.

3023: delayname already used

The *delayname* has already been used as a local delay. Choose another name.

3024: illegal loop counter value (not in 0..255)

The channel loop counter is a 8 bit counter and therefore may only be loaded with values in the range from 0 to 255.

3025: illegal channel for c function (not in 1..16)

For update, init and read functions the channel must be in the range from 1 to 16, or ’G’.

3026: delay not found

The *delayname* used in an update, init or read function is unknown.

3027: delay already used in update function

Do not use the same *delayname* in two different update functions to avoid inconsistent sets of delay data.

3028: channel already used in update function

Do not use delays from one channel in two different update functions to avoid inconsistent sets of delay data.

3029: illegal delay for update function

The *delayname* used in an update function is unknown.

4000: illegal simulation command

Valid simulation commands are *WR*, *EXT, UPDATE, SWRES* and *SWTR.*

4001: illegal time value (format)

The time value of a simulation command could not be interpreted by the compiler. Use real or integer numbers only.

4002: illegal time value (range)

The time value of a simulation command, scaled with *UNIT*, exceeds the range 0 to 26.84 sec.

4003: illegal EXT parameter

The level parameter of the *EXT* command could not be interpreted by the compiler. Use ’L’ or ’H’ only.

4005: illegal SWRES parameter

The level parameter of the *SWRES* command could not be interpreted by the compiler. Use ’L’ or ’H’ only.

4006: illegal delay value (format)

The delay value of the *WR* command could not be interpreted by the compiler. Use real or integer numbers only.

4007: illegal delay value (range)

The delay value of the *WR* command, scaled with *UNIT*, exceeds the range 0 to 26.84 sec.

4008: illegal SWFL parameter (level)

The level parameter of the *SWFL* command could not be interpreted by the compiler. Use ’L’ or ’H’ only.

4009: illegal SWFL parameter (flag)

The flag parameter of the *SWFL* command could not be interpreted by the compiler. Flags in the range 1 to 7 are allowed only.

4010: illegal channel

Channel must be in the range 1 to 16. In addition, ’G’ is also legal to indicate ’global’.

9999: internal error

Please report this error message along with the source file to dSPACE.

4.2.4 Warnings

5000: delay too short

DELAY command: Delay constants should not be smaller than 10 ticks. This is the time the channel controller needs to proceed to the next state. Smaller delay constants might lead to unexpected signal waveforms.

5001: delay too short

SIMULATE/WR command: Delay constants should not be smaller than 10 ticks. This is the time the channel controller needs to proceed to the next state. Smaller delay constants might lead to unexpected signal waveforms.

5002: unit changed

The base time unit has been redefined. This could lead to unexpected results, if channels defined later in the source file still rely on the first unit declaration. Therefore the *UNIT* command should only be used once at the beginning of a source file.

4.3 Delay Update Modes

On the DS5101 delay constants can be changed while the board is running. Depending on the application, the user may wish the changes to become effective under different conditions:

- changes become effective only at selected states (synchronous update mode)

- changes become effective immediately, even for the state which is being executed (immediate update mode)

One of these modes has to be selected for each channel.

4.3.1 Synchronous Update Mode

This mode is used if changes in a set of delay values shall become effective only at special program state.

Example: A PWM generator needs at least two delays for a complete cycle (low and high period). Those delays always have to be changed simultaneously in a way that the sum of both remains constant, and changes may only be done at the beginning of a cycle. If a change would only affect the second part of a period, the output signal would change its phase. This would be fatal, if this channel was one of three channels providing control signals with a 120 degree offset scheme for a three phase electrical motor.

In the DS5101 source file, all states in which delay changes may become effective must contain the *SYNCEN* command. Do not use the *IMMEDIATE* command, because it has priority over the *SYNCEN* command.

The PHS bus master must use the update functions from the C output file to write new data to the next available delay buffer. Changes will become effective when the channel controller encounters the next *SYNCEN* command.

4.3.2 Immediate Update Mode

This mode is used if changes in a set of delay values shall become effective immediately.

Example: A free running signal generator needs to be actualized in a way that changes become effective immediately. This may be useful if a very long pulse has been started, but the PHS bus master has decided to shorten or terminate this pulse at once.

In the DS5101 source file, the *IMMEDIATE* command must be given as the first command within a channel definition. In this mode, the channel controller permanently accepts new delay buffers and reloads the new delay value.

The PHS bus master must use the update functions from the C output file to write new data to the next available delay buffer. Changes will become effective immediately.

5 Hardware Reference

5.1 PHS-bus Interface Description

The DS5101 appears on the PHS-bus as a set of 32-bit random access registers. The base address of this register set (board address) must be manually set by the DIP-switches (S1) located on the left side of the board (refer to Figure 5.3.1). For board versions DS5101-04 and later, the DIP-switches have been replaced by a rotary switch which directly shows the hexadecimal base address.
This base address is set in increments of 16 according to Table 5.1.1. The labels shown in the table correspond to the numbers printed on the DIP-switches on the DS5101. The default PHS-bus base address of the DS5101 is D0H.

Base addr.	S1.1	S1.2	S1.3	S1.4	S1 (5101-04)
00H	On	On	On	On	0
10H	On	On	On	Off	1
20H	On	On	Off	On	2
30H	On	On	Off	Off	3
40H	On	Off	On	On	4
50H	On	Off	On	Off	5
60H	On	Off	Off	On	6
70H	On	Off	Off	Off	7
80H	Off	On	On	On	8
90H	Off	On	On	Off	9
A0H	Off	On	Off	On	A
B0H	Off	On	Off	Off	B
C0H	Off	Off	On	On	C
D0H	Off	Off	On	Off	D
E0H	Off	Off	Off	On	E
F0H	Off	Off	Off	Off	F

Table 5.1.1. Base address setting.

Before setting the base address check that this value is not occupied by another PHS-bus peripheral board. The setup program delivered with the PHS-bus master (processor board) can be used to obtain a list of all connected peripheral boards and their corresponding base addresses.

The DS5101 contains 12 registers to access and control the waveform generation modules. To access a particular register the offset given must be added to the board base address. Table 5.1.2 shows the address map of the DS5101.

Offset	Access	Name	Function
00H	RD/WR	MDR	Module data register
01H	RD/WR	-	Not used, reserved
02H	RD/WR	-	Not used, reserved
03H	RD/WR	ISP	FPGA update register (5101-04 only)
04H	RD/WR	FLR	Flag register
05H	RD/WR	STP	Setup register
06H/07H	RD/WR	ICU	Interrupt control unit
08H	RD/WR	-	Not used, reserved
09H	RD/WR	RR	Reset register
0AH	RD/WR	IR	Interrupt Register
0BH	RD/WR	IMR	Interrupt Mask Register
0CH	RD/WR	ADR	Address register, Channel select
0DH	WR	STR	Software Trigger register
0EH	RD	RBR	Readback register
0FH	RD	IDR	Identification register

Table 5.1.2. PHS-bus register address map.

The alignments of the registers within the 32-bit word are shown in Figure 5.1.1. All unused bits of a 32-bit word (in gray shaded areas) are undefined when read and should be zero when written.

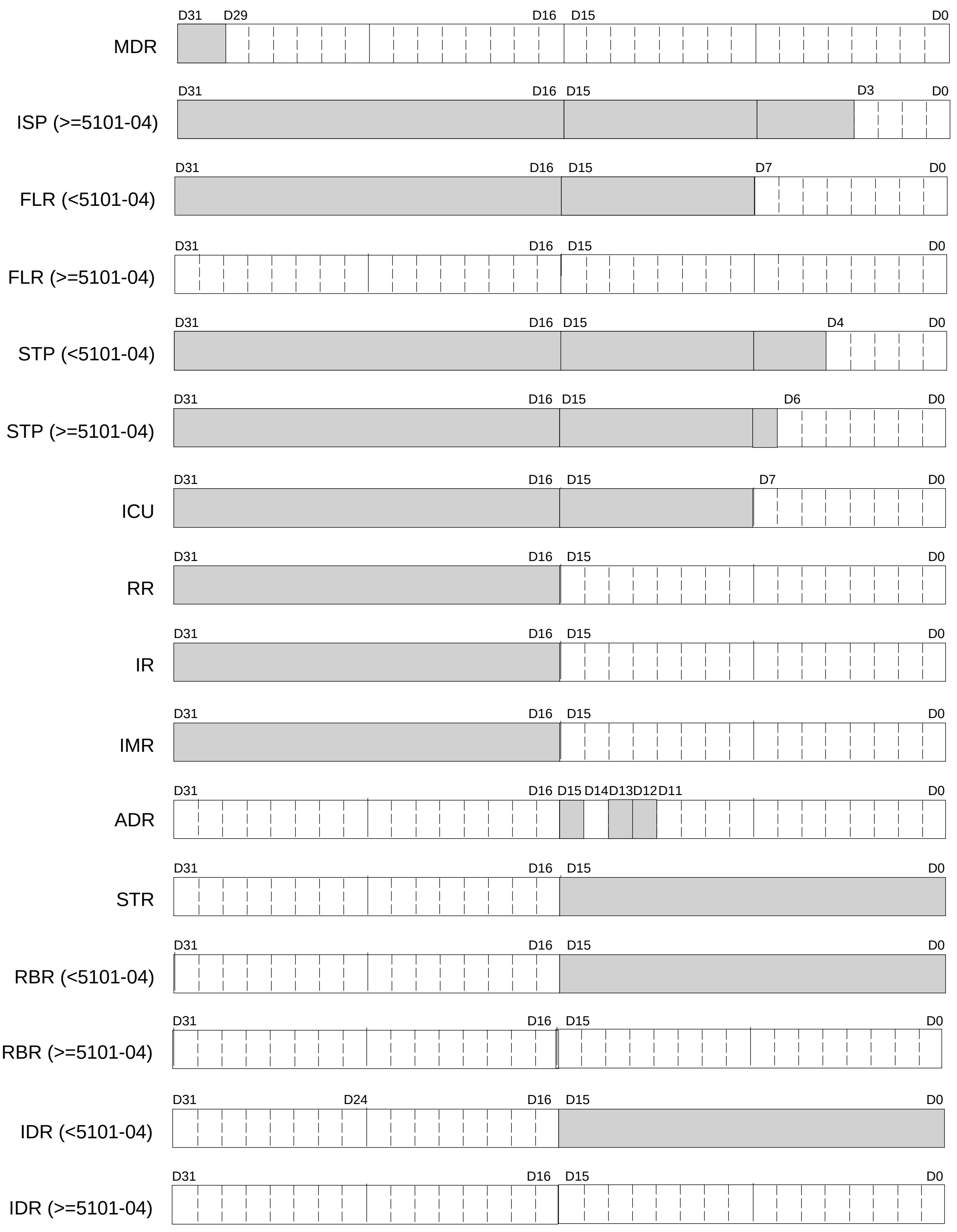


Figure 5.1.1. PHS-bus register alignment.

5.1.1 Module data register (MDR)

The Module data register (MDR) is a 30-bit (board versions DS5101-04 and later: 32-bit) wide read/write register used to access the state and delay buffer memory of the modules. Read and write operations on the data register are always performed on the memory location currently selected by the address register of the chosen channel.

For board versions before DS5101-04, Bit 30 and 31 of the MDR are zero when read.

Figure 5.1.2 shows the format of the Module data register.

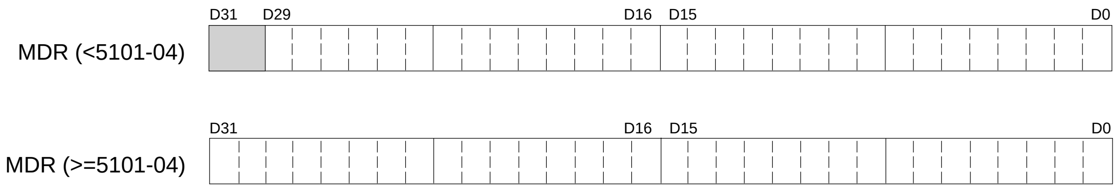


Figure 5.1.2. Module data register (MDR).

5.1.2 ISP register (ISP)

The ISP register is only available on board versions DS5101-04 and later.
The ISP register is a 4 bit read/write register which allows access to the JTAG port of the FPGA boot device. It can be used for in system programming / updating of the FPGA functionality.
Never try to access this register directly, only use dSPACE tools to perform FPGA updates!

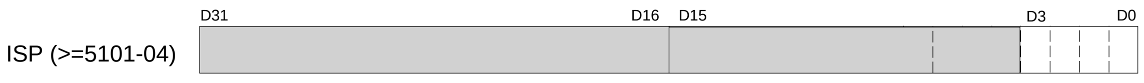


Figure 5.1.3. ISP register (ISP).

Bit	Name	Function
0	TDI	TDI port of the FPGA boot device
1	TDO	TDO port of the FPGA device, read only. The data path is a chain through the boot device and the FPGA itself: TDI(ISP register) - TDI(boot device) - TDO(boot device) - TDI(FPGA) - TDO (FPGA) - TDO(ISP register)
2	TMS	TMS port of the FPGA boot device
3	TCK	TCK port of the FPGA boot device
4 • 31	protection	Writing to bits 3..0 of the ISP register is only possible, when bits 31..4 are filled with a protection pattern.

Table 5.1.3. ISP register (ISP).

5.1.3 Flag register (FLR)

The flag register is a 8 bit (board versions DS5101-04 and later: 32 bit) read/write register containing 7 (31) flags and the UPDATE-ENABLE bit. The flags are connected to all modules and can be used for conditional branches within the program executed by the controller. Please refer to section 5.2.1.3.12. (*IF / ELSE*). The UPDATE-ENABLE bit (in conjunction with the address register AR) is used to control the swinging buffer controllers.

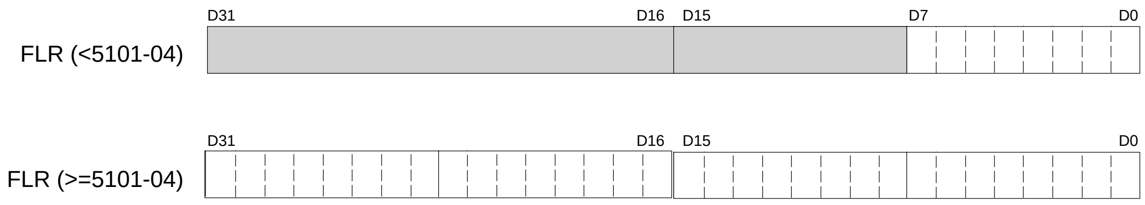


Figure 5.1.4. Flag register (FLR).

Bit	Name	Function
0 • 6	FLAG1 • FLAG7	Writing a one sets the respective FLAG, writing a zero resets the corresponding FLAG. bit0 FLAG1 bit1 FLAG2 bit2 FLAG3 bit3 FLAG4 bit4 FLAG5 bit5 FLAG6 bit6 FLAG7
7	UPDATE- ENABLE	Enable update. A rising edge at UPDATE-ENABLE causes the swinging buffer controller to proceed to the next delay buffer if - the address section of the address register contains a value of binary ’000’ in bits AR11..9 and - the corresponding channel select bit in the address register is set. (refer to section 5.1.9)
8 • 31	FLAG8 • FLAG31	Writing a one sets the respective FLAG, writing a zero resets the corresponding FLAG. Only available on board versions DS5101-04 and later! bit8 FLAG8 bit9 FLAG9 ... ... bit31 FLAG31

Table 5.1.4. Flag register (FLR).

5.1.4 Setup register (STP)

5.1.4.1 Software setup

The setup register (STP) is a 5 bit (board versions DS5101-04 and later: 7 bit) read/write register used to control several operations of the DS5101. It contains the PHS-bus interrupt line select field, the clock select line and the pull-up level select line. If the setup is performed manually by jumpers the STP register can only be read.

For board versiond DS5101-04 and above, the FPGA boot status and reset are available.

Figure 5.1.5 shows the format of the STP register.

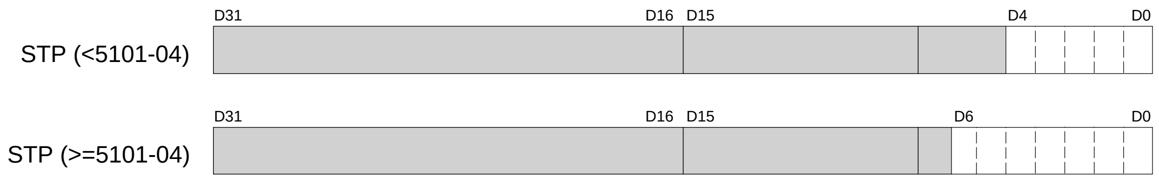


Figure 5.1.5. Setup register (STP).

Bit	Name	Function
0 ● 2	INTSEL	Interrupt line select field. INTSEL selects one of eight PHS-bus interrupt lines to be used for extended interrupts via the ICU (refer to section 5.1.5). The format of INTSEL is as follows: 000 line 0 001 line 1 010 line 2 011 line 3 100 line 4 101 line 5 110 line 6 111 line 7
3	CLKSEL	Clock select flag. CLKSEL = 0 selects the on-board 40 MHz clock. CLKSEL = 1 selects the external clock input. On power-up CLKSEL is reset.
4	PULLUP	PULLUP selects the level of the I/O pins while the controllers are in reset or programmed to input mode. PULLUP = 0 connects the pull-up resistors to GND. PULLUP = 1 connects the pull-up resistors to VCC. On power-up PULLUP is set.

5	FPGARES	Only available on board versions DS5101-04 and later! FPGARES is an additional reset only effective for the FPGA section of the board. Only needed for diagnostic purposes, do not use!
		FPGARES = 0: no operation (default)
		FPGARES = 1: reset FPGA section

6	FPGADONE	Only available on board versions DS5101-04 and later! FPGADONE is read only and gives the boot state of the FPGA. Only needed for diagnostic purposes. FPGADONE = 0: FPGA not booted correctly (not ready yet or boot device content is corrupted) FPGADONE = 1: FPGA booted correctly

Table 5.1.5. Setup register (STP).

5.1.4.2 Manual setup

All board setups programmable in the STP register may be performed manually by inserting jumpers. This requires that the STP register is put into manual setup mode by inserting jumper J1.1. In manual setup mode the STP cannot be written but may be read to check for correct jumpering. Table 5.1.6 shows the function of the various setup jumpers and Figure 5.3.1 shows the jumper locations on the DS5101.

Jumper	Function	Inserted	Removed
J1.1	Setup mode	manual	programmable
J1.2 J1.3 J1.4	Interrupt line select 0 (LSB) Interrupt line select 1 Interrupt line select 2 (MSB)	binary 0 binary 0 binary 0	binary 1 binary 1 binary 1
J1.5	Clock select	on-board	external
J1.6	Pull-up select	to GND	to VCC

Table 5.1.6. Manual setup.

Always remove all setup jumpers if switching to programmable setup mode (removing J1.1). Failing to do so may result in hardware damage.

5.1.4.3 Extended Manual setup

For board versions DS5101-04 and later, additional setup jumpers have been added. They allow extended pullup or pulldown features for each channel individually.

There are 36 additional jumpers, labeled
1A, 1B, 2A, 2B, ... , 16A, 16B, S1, S2, S3, S4.

When S1 is open, the pullup function is defined by software setup or manual setup as described in the two preceeding chapters. This behaviour is compatible to board versions earlier than DS5101-04.

When S1 is closed, each channel (x=1..16) can be configured to behave individually in input mode. Software setup and manual setup have no effect in this case.

S2, S3 and S4 do not yet have any function.

There are 4 modes, selectable through jumpers xA and xB for each channel. Table 5.1.7 shows the modes and Figure 5.3.1 shows the jumper locations on the DS5101.

Switch / Jumper			Function
S1	xA	xB	
open	dont care	dont care	Pullup function according to software setup or manual setup
closed	open	open	no Pulldown or Pullup
closed	open	closed	Pullup 10K to VCC
closed	closed	open	Pulldown 10K to GND
closed	closed	closed	Pull 10K to current level (hold function)

Table 5.1.7. Extended manual setup.

5.1.5 Interrupt control unit (ICU)

The DS5101 contains a slave interrupt control unit (ICU) which expands one of the eight master interrupt lines of the PHS-bus to eight slave interrupt inputs. The slave ICU supports two interrupts from the DS5101. The first interrupt input is driven by the DS5101 interrupt register (IR5101) which contains a priority encoder of the 16 interrupt inputs from each channel. The second interrupt is driven by the external reset input.

The ICU consists of a standard interrupt controller chip (PIC) compatible to the INTEL 8259A, a buffer and a line selection logic to connect it to the PHS-bus. The master interrupt line used by the ICU for interrupt expansion is determined by the interrupt line select field in the setup register (refer to section 5.1.4). Each DS5101 board requires a separate interrupt line. Polling of the interrupt input is also possible by programming the ICU to level triggered mode and reading the slave-ICU’s interrupt request register (IRR). A one in an IRR-bit signals that the corresponding interrupt input is active (high).

Note that even for polled operation the slave ICU must be correctly initialized to obtain access to the IRR. Since the PIC’s data lines are buffered it must be programmed to buffered slave mode via IW4. Refer to the PIC data-sheets for proper initialization (e.g. NEC V-series Data Book µpD71059 interrupt control unit).

The mapping of the interrupt inputs of the slave ICU is shown in Table 5.1.8.

PIC input	Function
INTP 0	Output of the interrupt register (IR), active if at least one interrupt from the modules is requested
INTP 1	External reset input.
INTP 2 • INTP 7	Not used. Connected to GND.

Table 5.1.8. Interrupt input mapping.

5.1.6 Interrupt mask register (IMR)

The interrupt mask register (IMR) is used to enable or disable the flags from the interrupt register (IR) for interrupt generation to PHS-bus. A one enables the respective channel, a zero disables the channel. Interrupt line 0 from the Interrupt Control Unit is activated if at least one channel has executed a PHSINT command and the respective channel is enabled in the interrupt mask register. When the interrupt is served the respective interrupt register bit must be reset by writing a ’1’ to this bit.

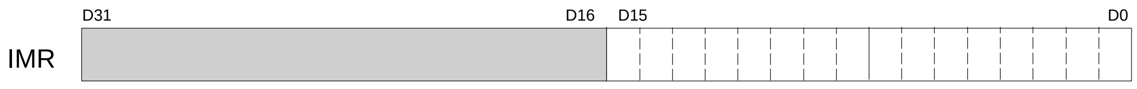


Figure 5.1.6. Interrupt mask register (IMR).

Table 5.1.9 shows the format of the Interrupt mask register (IMR).

Bit	Name	Function
0 • 15	IMR1 • IMR16	Interrupt mask register. A one enables the interrupt, a zero disables the interrupt. The bits are described as follows: bit0 Channel 1 bit1 Channel 2 bit2 Channel 3 bit3 Channel 4 bit4 Channel 5 bit5 Channel 6 bit6 Channel 7 bit7 Channel 8 bit8 Channel 9 bit9 Channel 10 bit10 Channel 11 bit11 Channel 12 bit12 Channel 13 bit13 Channel 14 bit14 Channel 15 bit15 Channel 16

Table 5.1.9. Interrupt mask register (IMR).

5.1.7 Reset register (RR)

The reset register (RR) is a 16 bit read/write register used to reset the waveform generation controllers. After reset the controllers are set to input mode.

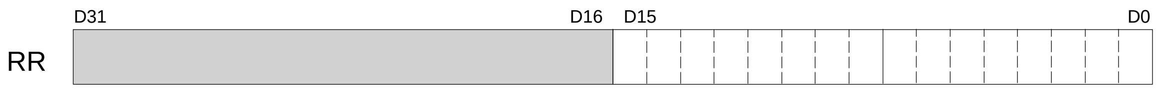


Figure 5.1.7. Reset register (RR).

Bit	Name	Function
0 • 15	RES1 • RES16	Reset channel. RESx = 1 resets the corresponding channel. RESx = 0 restarts the respective channel. bit0 Channel 1 bit1 Channel 2 bit2 Channel 3 bit3 Channel 4 bit4 Channel 5 bit5 Channel 6 bit6 Channel 7 bit7 Channel 8 bit8 Channel 9 bit9 Channel 10 bit10 Channel 11 bit11 Channel 12 bit12 Channel 13 bit13 Channel 14 bit14 Channel 15 bit15 Channel 16 On power-up RESx = 1. An active external reset input or the PHS-bus IOERR line sets all reset bits.

Table 5.1.10. Reset register (RR).

5.1.8 Interrupt register (IR)

The interrupt register (IR) is used to latch the interrupt requests from the modules (see compiler command PHSINT). The interrupt register activates the interrupt line0 from the Interrupt Control Unit if at least one channel has requested an interrupt. The interrupt register can be read to detect which channels have caused the interrupt.

Using a read-modify-write operation to reset the pending interrupts could result in the loss of interrupts, which occurred between the read and the write. Therefore, when the interrupt is served the respective interrupt flag must be reset by writing a ’1’ to the corresponding bit in the IR register. Writing a ’0’ does not affect the interrupt register bits.

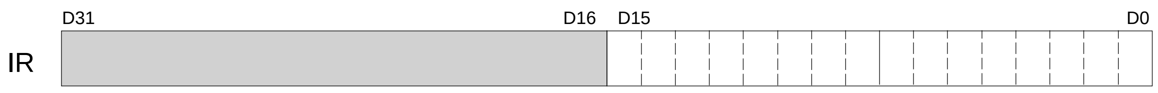


Figure 5.1.8. Interrupt register (IR).

The interrupt register is reset on power-up, PHS-bus I/O error and on active external reset signal.

Table 5.1.11 shows the format of the interrupt register.

Bit	Name	Function
0 • 15	IR1 • IR16	Interrupt request field. When read, a ’1’ signals a pending interrupt request. When written, a ’1’ resets the request bit. A ’0’ has no effect. The bits are described as follows: bit0 Channel 1 bit1 Channel 2 bit2 Channel 3 bit3 Channel 4 bit4 Channel 5 bit5 Channel 6 bit6 Channel 7 bit7 Channel 8 bit8 Channel 9 bit9 Channel 10 bit10 Channel 11 bit11 Channel 12 bit12 Channel 13 bit13 Channel 14 bit14 Channel 15 bit15 Channel 16

Table 5.1.11. Interrupt register (IR)

5.1.9 Address register (AR)

The address register (AR) is a 30-bit wide read/write register containing a 12 bit address field, an autoincrement enable flag and a 16 bit channel select field.

The address register has an autoincrement mode which can be used for block transfers between the PHS-bus master and the module memories. In this case the address is incremented after a read or write operation to the MDR register.

The channel select field selects the channel accessed by the following read or write operations to the data registers. If several channels should be updated with the same data only one write operation is necessary because more than one channels can be written to simultaneously. The user has to take care that for read operations only one channel is selected.

Writing the Address register resets the UPDATE-ENABLE flag in the flag register.

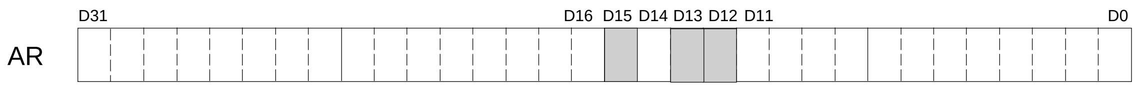


Figure 5.1.9. Address register (AR).

Bit	Name	Function
0 • 11	ADDR	Address register. ADDR is a 12 bit autoincrement address register used to select the module memory address of the next read or write operation via the module data register (MDR). On power-up ADDR is undefined.
14	ENINC	Autoincrement enable bit. ENINC = 1 enables autoincrement mode. ENINC = 0 disables autoincrement mode. On power-up ENINC is undefined.

Table 5.1.12. Address register (AR).

Bit	Name	Function
16 • 31	CHSEL1 • CHSEL16	Channel select field. CHSEL selects the channels accessed on the following read or write operations via the module data register (MDR). If a bit is set the specified channel is selected. If a bit is reset the respective channel is not selected. The format of CHSEL is as follows: Bit 16 Channel 1 Bit 17 Channel 2 Bit 18 Channel 3 Bit 19 Channel 4 Bit 20 Channel 5 Bit 21 Channel 6 Bit 22 Channel 7 Bit 23 Channel 8 Bit 24 Channel 9 Bit 25 Channel 10 Bit 26 Channel 11 Bit 27 Channel 12 Bit 28 Channel 13 Bit 29 Channel 14 Bit 30 Channel 15 Bit 31 Channel 16

Table 5.1.13. Address register (AR) (continued)

A common problem in real time systems is data consistency, if one unit (here the DSP as PHS bus master) generates sets of data, and another unit (here the channel controller) has to use them. If DSP and controller used the same data buffer RAM, the channel controller would almost inevitably get part of its data changed by the DSP while executing the program, thus leading to unexpected and inpredictable results.

The DS5101 takes another approach. States are written to the state buffer once during initialization and will not change during program execution. In order to change states the channel has to be reset.

Delays are stored in three delay buffers of the same size. One buffer is always available for the DSP to write new delay data sets. One buffer may contain data which has been written by the DSP, but has not yet been used by the channel controller. The last buffer is always available for the channel controller to work with.

The management of the delay buffers is done by the Swinging Buffer Controller (SBC), which is implemented once for each channel. The three delay buffers are named A, B and C. The state of the SBC can be described as ’CBA’, which means
C is available for the DSP,
B contains new data, and
A is used by the channel controller.
At state ’C0A’, there is no new data available.

After reset, the SBC is in state ’C0A’. All three buffers A, B and C must be initialized to contain the same basic set of data.

The DSP writes new data to buffer C. After setting the UPDATE ENABLE bit in the flag register with the address register set properly, the SBC will change to state ’BCA’.

When the channel controller encounters a SYNCEN command, or if it is in IMMEDIATE update mode, the SBC will change to state ’B0C’, so that the new data in buffer C becomes effective.

Of course, neither the DSP nor the channel controller know about the actual SBC state. They do not need to, because the SBC provides access to the correct delay buffer for both of them.

This is done by using the address bits A11..9 of the address register AR as mode bits. They decide whether a direct access with bypassing of the SBC is desired, or if the actual DSP buffer (’C’ in state ’CBA’) is available.

Table 5.1.14 shows the function of these address bits.

A11..9	Mode	Function
000	DSP buffer	Address bits 10 and 9 are replaced by the SBC with the address bits for the buffer, which is available for updates. This mode is used in the update C functions generated by the compiler.
001	last buffer	Address bits 10 and 9 are replaced by the SBC with the address bits for the buffer, which contains the newest data. This is the buffer desribed as ’ready’ (buffer ’B’ in state ’CBA’, or the buffer used by the channel controller, if no ’ready’ buffer is available (buffer ’A’ in state ’C0A’). This mode is used in the read C functions generated by the compiler.
010	reset SBC	The SBC is reset to state ’C0A’.
011	-	unused
1xx	direct	Address bits 10 and 9 from the address register are fed directly to the channel dual port RAM. This mode is used in the initialization C functions generated by the compiler to access all state and delay buffers directly. Do not use this mode while the channel controller is running.

Table 5.1.14. Address Register Modes

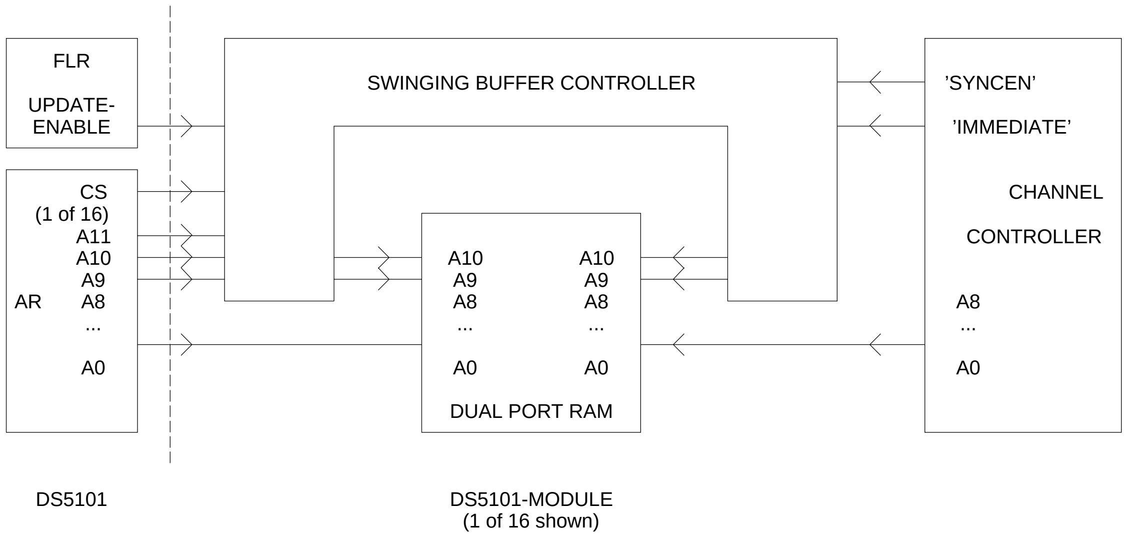


Figure 5.1.10. Swinging buffer controller.

5.1.10 Software trigger register (STR)

The Software trigger register is a 16 bit write only register used to trigger the waveform generation controller. The controller can be programmed to wait until a software trigger is executed.

Figure 5.1.11 shows the format of the software trigger register.

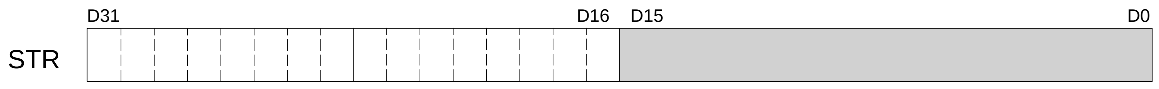


Figure 5.1.11. Software trigger register (STR).

Bit	Name	Function
16 • 31	STR1 • STR16	Software trigger channel 1 to 16 . A one triggers the controller on the respective channel, a zero has no effect. The bits are described as follows: Bit 16 Channel 1 Bit 17 Channel 2 Bit 18 Channel 3 Bit 19 Channel 4 Bit 20 Channel 5 Bit 21 Channel 6 Bit 22 Channel 7 Bit 23 Channel 8 Bit 24 Channel 9 Bit 25 Channel 10 Bit 26 Channel 11 Bit 27 Channel 12 Bit 28 Channel 13 Bit 29 Channel 14 Bit 30 Channel 15 Bit 31 Channel 16

Table 5.1.15. Software trigger register (STR).

5.1.11 Readback register (RBR)

The readback register is a 16 bit (DS5101-04 and later: 32 bit) read only register used to monitor the I/O signals. A high-level at the I/O pin sets the respective flag in the readback register, a low-level resets the flag.

For board versions DS5101-04 and later, the RBR also contains the 16 bit firmware version of the FPGA. It is given in a 8.8 format, which means V1.0 is read as 0x0100.

Figure 5.1.12. shows the format of the readback register.

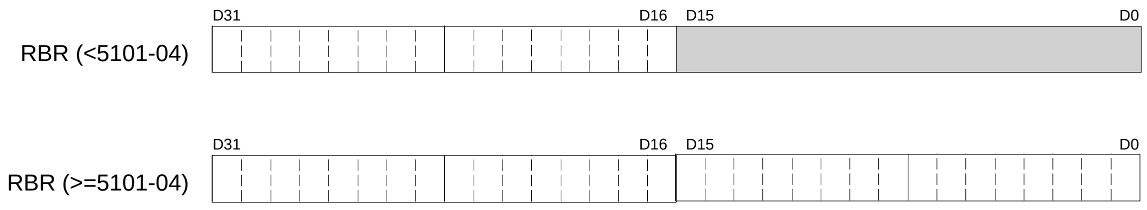


Figure 5.1.12. Readback register (RBR).

Bit	Name	Function
0 ● 15	FV0 ● FV15	Only available on board versions DS5101-04 and later FPGA firmware version/revision, given in 8.8 format

16 ● 31	RB1 ● RB16	Readback register channels 1 to 16. The format of RB is as follows Bit 16 Channel 1 Bit 17 Channel 2 Bit 18 Channel 3 Bit 19 Channel 4 Bit 20 Channel 5 Bit 21 Channel 6 Bit 22 Channel 7 Bit 23 Channel 8 Bit 24 Channel 9 Bit 25 Channel 10 Bit 26 Channel 11 Bit 27 Channel 12 Bit 28 Channel 13 Bit 29 Channel 14 Bit 30 Channel 15 Bit 31 Channel 16

Table 5.1.16. Readback register (RBR).

5.1.12 ID register (IDR)

Each slave board of the dSPACE modular hardware family contains an identification register at address offset 0FH. This register is used by the monitor program coming with the PHS-bus master to check hardware-software integrity. The IDR is a read-only register and contains an eight bit ID-number in data bits D24 to D31 and an eight bit PCB version number in data bits D16 to D23. The ID-number of the DS5101 board is E2H.

For board versions DS5101-04 and later the lower bits of the ID register are also used:

D15..9 are set to low
D8..1 gives the firmware version of the PHS bus interface PAL
D0 is set to high to indicate a non-PHS++ type board.

So, a DS5101-04 with firmware version 1 will identify itself with 0xE2040003. A DS5101-02 would read as 0xE202xxxx.

5.2 Module Description

The module (DS5101M) contains two waveform generation channels. Each channel consists of a channel controller, a swinging buffer controller and a dual port RAM. The controller can execute an especially designed programming language to generate waveforms comfortably. This language is described in section 5. of this document.

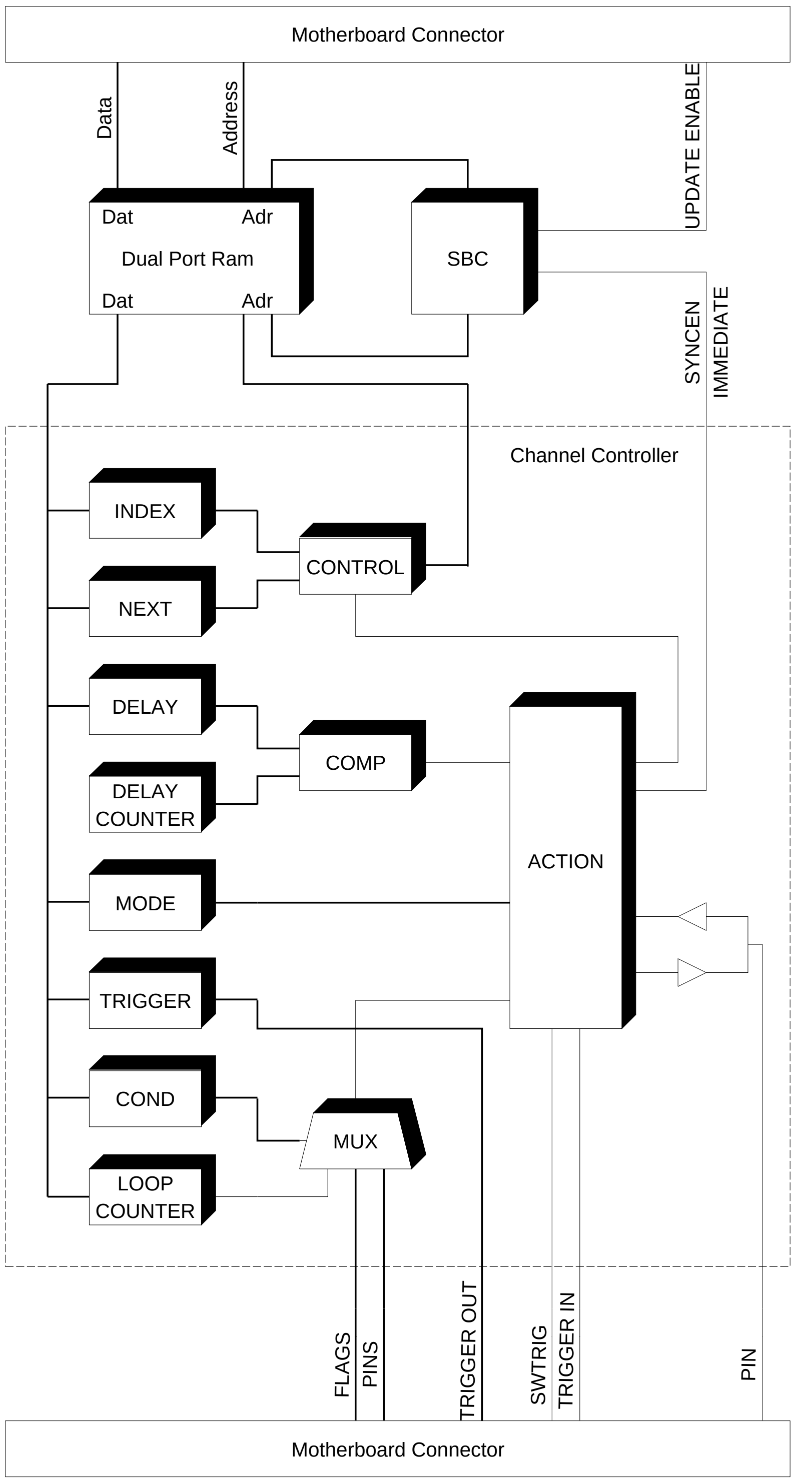


Figure 5.2.1 Block diagram of one waveform generation channel.

The controller consists of a 30 bit counter, a compare unit, a control unit, an action unit and a set of registers. After reset the control unit reads the first set of data into the respective registers and starts the counter. If the counter value matches the contents of the delay register the action register will be triggered. The action register sets the output to the programmed value and starts the control unit to fetch the next set of data. The action register can also be triggered by the external pin (input mode), by an trigger input from the other channels (wait) or from a software trigger by the PHS bus master (wait).

The cond register selects one of 24 inputs for the next conditional branch decision. The inputs are the 16 channels, 7 flags controlled by the PHS-bus master or the zero output of a loop counter.

The RAMs are divided into four buffers: the state buffer and three delay buffers. The delay buffers are managed by a Swinging Buffer Controller (SBC), which is described in detail in the previous section ’Address Register (AR)’ (see figure 6.2.2).

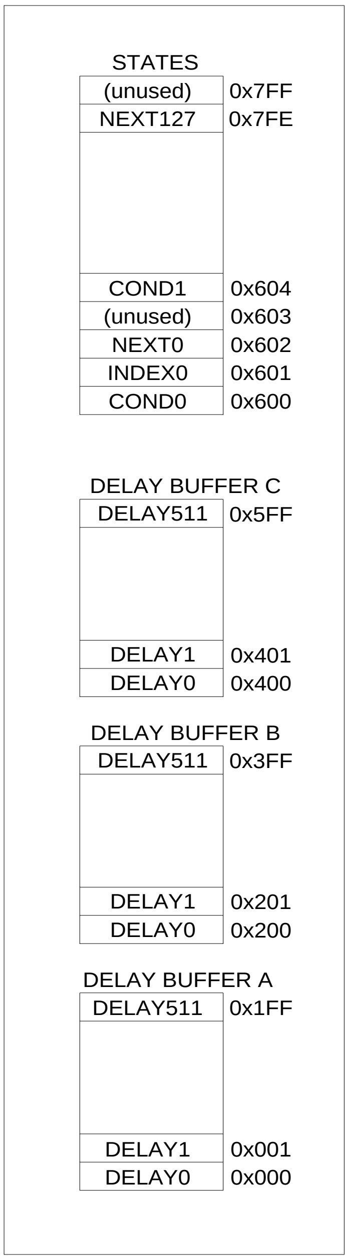


Figure 5.2.2. Module memory map.

5.2.1 Command word description

The Command word consists of three 31 or 30 bit words: the CONDITION command word, the INDEX command word and the NEXT command word. The CONDITION command word contains the condition select word, the SYNCEN flag, the PHS-bus interrupt flag, the mode words and the trigger word. The INDEX command word covers the two delay addresses and part of the loop counter. The NEXT command word contains the next address location for the succeeding set of data, the update flag and the rest of the loop counter.

5.2.1.1 Condition command word

The CONDITION command word consists of the condition field, control flags, mode fields and the trigger value. For board versions DS5101-04 and later, the COND field is 6 bits wide instead of 5, with the additional bit COND5 added at bit position 30 to maintain compatibility with earlier board versions.

	D30	D29 .. D15	D14 .. D11	D10 .. D7	D6	D5	D4 .. D0
CONDITION	CON D5	TRIGGER	MODE if	MODE else	PHSINT	SYNCEN	COND4..0

Figure 5.2.3. Condition word format.

Bit	Name	Function
0 • 4 / 30	COND	Condition field selects one of 32 inputs for the if / else condition. The bits are described as follows: 000000 External pin channel 1 000001 External pin channel 2 000010 External pin channel 3 000011 External pin channel 4 000100 External pin channel 5 000101 External pin channel 6 000110 External pin channel 7 000111 External pin channel 8 001000 External pin channel 9 001001 External pin channel 10 001010 External pin channel 11 001011 External pin channel 12 001100 External pin channel 13 001101 External pin channel 14 001110 External pin channel 15 001111 External pin channel 16 010000 Flag 1 010001 Flag 2 010010 Flag 3 010011 Flag 4 010100 Flag 5 010101 Flag 6 010110 Flag 7 010111 TRUE 011000 Loop counter zero 011001 TRUE ... 011111 TRUE 100000 Flag 8 100001 Flag 9 100010 Flag 10

			100011 Flag 11
			100100 Flag 12
			100101 Flag 13
			100110 Flag 14
			100111 Flag 15
			101000 Flag 16
			101001 Flag 17
			101010 Flag 18
			101011 Flag 19
			101100 Flag 20
			101101 Flag 21
			101110 Flag 22
			101111 Flag 23
			110000 Flag 24
			110001 Flag 25
			110010 Flag 26
			110011 Flag 27
			110100 Flag 28
			110101 Flag 29
			110110 Flag 30
			110111 Flag 31
			111000 TRUE
			...
			111111 TRUE

Table 5.2.1. Condition command word.

Bit	Name	Function
5	SYNCEN	SYNCEN = 1 allows the Swinging Buffer Controller (SBC) to proceed to the next delay buffer, if new data are available. SYNCEN = 0 has no effect. On default SYNCEN = 0.
6	PHSINT	PHS-bus interrupt. PHSINT = 1 requests an interrupt to the PHS-bus master. PHSINT = 0 has no effect.
7 • 10	MODE else	Mode selected if condition is false. The bits are described as follows: Bits 9..7 select the different modes: 000 none input mode 001 fall input mode 010 rise input mode 011 both input mode 100 toggle output mode 101 reset output mode 110 set output mode 111 nop output mode Bit 10 selects delay or wait mode. Bit 10 = 0 selects delay mode, Bit 10 = 1 selects wait mode. In delay mode the controller is triggered when the counter matches the delay register. In wait mode the controller waits for an external trigger.
11 • 14	MODE if	Mode selected if condition is true. The bits are described in the MODE/else field.
15 • 29	TRIGGER	Trigger field. If a bit is set the respective channel is triggered. The trigger bits are different for each channel. Refer to section 5.2.1.4 for detailed information.

Table 5.2.2. Condition command word (continued).

5.2.1.2 Index command word

The INDEX command word contains two index fields for delay access.

	D29	D28..D24	D23..D15	D14	D13..D9	D8..D0
INDEX	LDCTR else	CTR4..0 else	INDEX else	LDCTR if	CTR4..0 if	INDEX if

Figure 5.2.4. Index word format.

Bit	Name	Function
0 • 8	INDEX if	Index address of the delay value if condition is true.
9 • 13	CTR4..0 if	5 low bits of the loop counter load value if condition is true.
14	LDCTR if	1: Load loop counter if condition is true. 0: no effect
15 • 23	INDEX else	Index address of the delay value if condition is false.
24 • 28	CTR4..0 else	5 low bits of the loop counter load value if condition is false.
29	LDCTR else	1: Load loop counter if condition is false. 0: no effect

Table 5.2.3. Index command word.

5.2.1.3 Next command word

	D29	D28	D27	D26	D25..24	D23..22	D21..15
NEXT	LDCTR else	DECCTR else	unused	IMMEDIATE else	unused	CTR6..5 else	NEXT else

D14	D13	D12	D11	D10..9	D8..7	D6..0
LDCTR if	DECCTR if	unused	IMMEDIATE if	unused	CTR6..5 if	NEXT if

Figure 5.2.5. Next word format.

Bit	Name	Function
0 • 6	NEXT if	Next address if condition is true.
7 • 9	CTR7..5 if	3 high bits of the loop counter load value if condition is true.
10	-	unused
11	IMMEDIATE if	IMMEDIATE = 1 selects the immediate update mode of the controller if the condition is true. The IMMEDIATE compiler command sets the IMMEDIATE bits for both condition true and false for the first state within a channel. Once IMMEDIATE has been set, it stays set until the channel is reset. IMMEDIATE = 0 has no effect.
12	-	unused
13	DECCTR if	1: Decrement loop counter if condition is true. 0: no effect
14	LDCTR if	1: Load loop counter if condition is true. 0: no effect
15 • 21	NEXT else	Next address if condition is false.
22 • 24	CTR7..5 else	3 high bits of the loop counter load value if condition is false.
25	-	unused

26	IMMEDIATE else	same as Bit 11 (IMMEDIATE/if).
27	-	unused
28	DECCTR else	1: Decrement loop counter if condition is false. 0: no effect
29	LDCTR else	1: Load loop counter if condition is false. 0: no effect

Table 5.2.4. Next command word.

5.2.1.4 Trigger field

The trigger field of the CONDITION command word is different for each channel. The following table shows the trigger bit locations for the respective channels. The left column shows the source channel. The first row shows the data locations in the command word. The other fields of the table show the destination channels.

Source Channel	D29	D28	D27	D26	D25	D24	D23	D22	D21	D20	D19	D18	D17	D16	D15
1	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2
2	16	15	14	13	12	11	10	9	8	7	6	5	4	3	1
3	2	1	16	15	14	13	12	11	10	9	8	7	6	5	4
4	2	1	16	15	14	13	12	11	10	9	8	7	6	5	3
5	4	3	2	1	16	15	14	13	12	11	10	9	8	7	6
6	4	3	2	1	16	15	14	13	12	11	10	9	8	7	5
7	6	5	4	3	2	1	16	15	14	13	12	11	10	9	8
8	6	5	4	3	2	1	16	15	14	13	12	11	10	9	7
9	8	7	6	5	4	3	2	1	16	15	14	13	12	11	10
10	8	7	6	5	4	3	2	1	16	15	14	13	12	11	9
11	10	9	8	7	6	5	4	3	2	1	16	15	14	13	12
12	10	9	8	7	6	5	4	3	2	1	16	15	14	13	11
13	12	11	10	9	8	7	6	5	4	3	2	1	16	15	14
14	12	11	10	9	8	7	6	5	4	3	2	1	16	15	13
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	16
16	14	13	12	11	10	9	8	7	6	5	4	3	2	1	15

Table 5.2.5. Trigger field.

Example : Channel 6 triggers channel 9 : D18 must be set.

5.2.2 Delay word description

The delay values are stored in the delay buffer RAM. The delay value is a 30 bit wide constant. The delay is given in 25ns ticks. The minimum delay value is 0AH (250ns) and the maximum delay value is 3FFFFFFFH (26.8s).

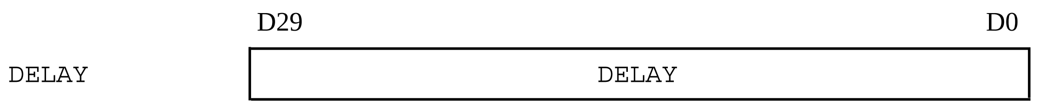


Figure 5.2.6. Delay word format.

5.2.3 Module insertion

Please note: Board versions DS5101-04 and later no not have modules any more. Instead, the functionality of 8 modules (= 16 channel) has been integrated into the base board logic FPGA.

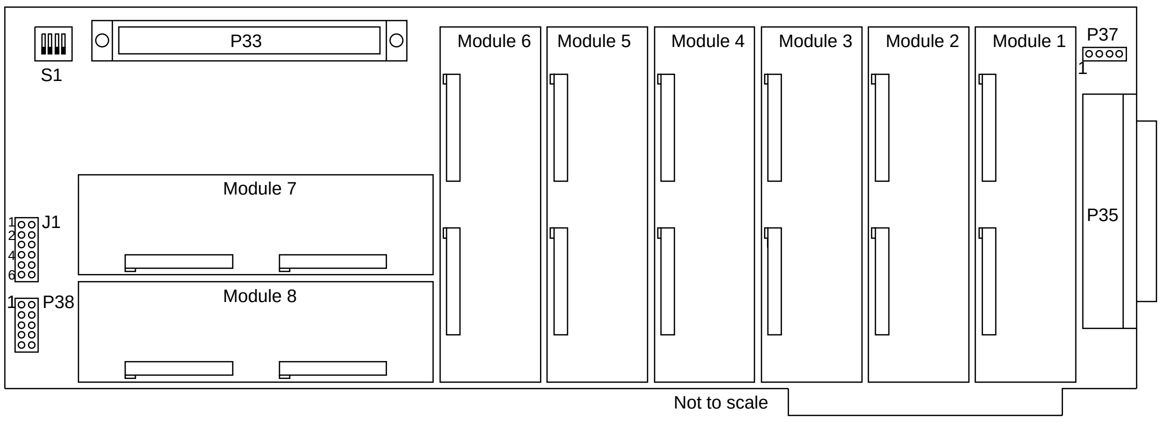
For earlier versions, installation of additional modules should be performed as follows :

- Turn off the host power supply.
- Remove the PHS-bus cable and the external cable from the DS5101.
- Remove the DS5101 board and put it on an ESD-protecting surface (e.g. the bag the board is delivered in).
- Insert the DS5101M module into the **next free** socket (refer to figure 5.3.1).
- Re-install the DS5101 board.

For checking the correct installation the PHS-bus master SED program can be used to display the number of installed channels. The SED program detects only consecutive inserted modules.

5.3 Connector Pin-outs

DS5101-02



DS5101-04

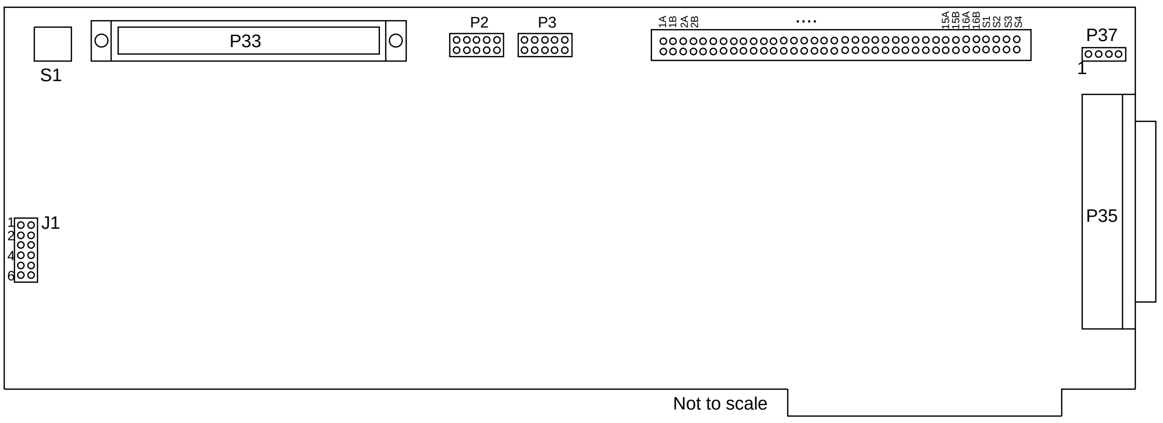


Figure 5.3.1. Module, jumper and connector locations.

5.3.1 PHS-bus connector (P33)

Pin	Signal	Pin	Signal
a1	GND	c1	GND
a2	D31	c2	D30
a3	D29	c3	D28
a4	D27	c4	D26
a5	D25	c5	D24
a6	D23	c6	D22
a7	D21	c7	D20
a8	D19	c8	D18
a9	D17	c9	D16
a10	D15	c10	D14
a11	D13	c11	D12
a12	D11	c12	D10
a13	D9	c13	D8
a14	D7	c14	D6
a15	D5	c15	D4
a16	D3	c16	D2
a17	D1	c17	D0
a18	A7	c18	A6
a19	A5	c19	A4
a20	A3	c20	A2
a21	A1	c21	A0
a22	INT7	c22	INT6
a23	INT5	c23	INT4
a24	INT3	c24	INT2
a25	INT1	c25	INT0
a26	CAS2	c26	/INTACK
a27	CAS0	c27	CAS1
a28		c28	/IOERR
a29	/IORD	c29	/IOWR
a30		c30	
a31		c31	
a32	GND	c32	GND

Table 5.3.1. PHS-bus connector pin-out

5.3.2 I/O connector (P35)

Pin	Signal
1	GND
2	GND
3	GND
4	GND
5	GND
6	GND
7	GND
8	GND
9	GND
10	GND
11	GND
12	GND
13	GND
14	GND
15	GND
16	GND
17	GND
18	GND
19	GND

Pin	Signal
20	Channel 1
21	Channel 2
22	Channel 3
23	Channel 4
24	Channel 5
25	Channel 6
26	Channel 7
27	Channel 8
28	Channel 9
29	Channel 10
30	Channel 11
31	Channel 12
32	Channel 13
33	Channel 14
34	Channel 15
35	Channel 16
36	GND
37	Ext. Reset

Table 5.3.2. I/O connector pin-out.

The I/O connector is a 37-pin female SUB-D connector located on the rear bracket of the DS5101 (refer to figure 5.3.1). The connector provides signal groups as shown in table 5.3.3. A mating 37-pin male SUB-D connector is delivered with the board.

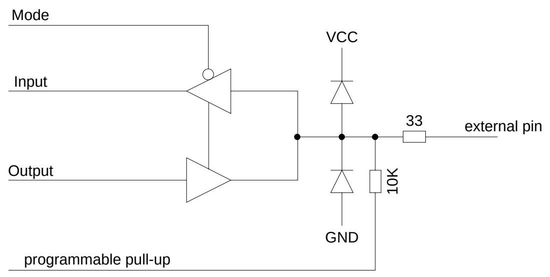
Name	Function
Channel 1 - 16	Digital I/O for waveform outputs or trigger inputs.
Ext. Reset	External reset input, active high.
GND	Return lines. Internally connected to System GND.

Table 5.3.3. I/O connector signal description.

All outputs are protected against temporary external voltages up to ±15V.

Figure 5.3.2 shows the interface circuit of the digital I/O lines and figure 5.3.3 shows the interface circuit of the external reset input.

DS5101-02:



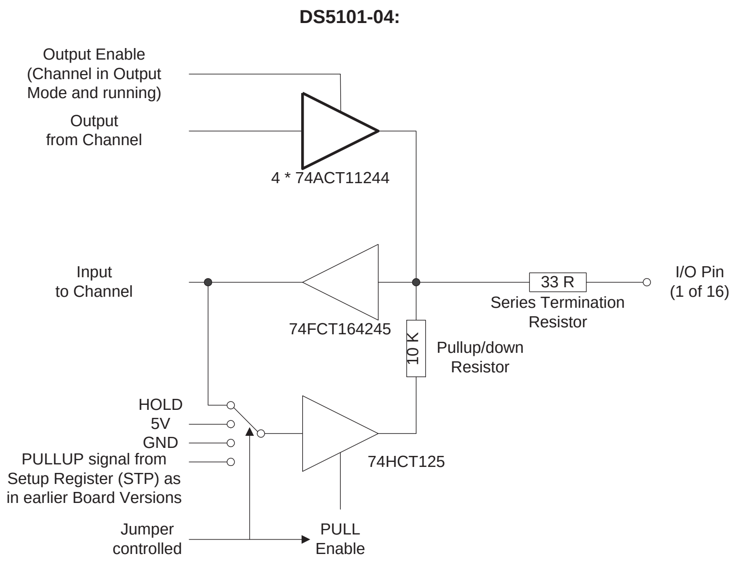


Figure 5.3.2. Interface circuit of the digital I/O lines.

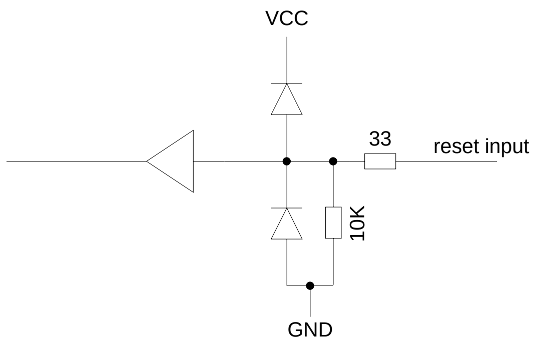


Figure 5.3.3. Interface circuit of the external reset input.

5.3.3 Module connector

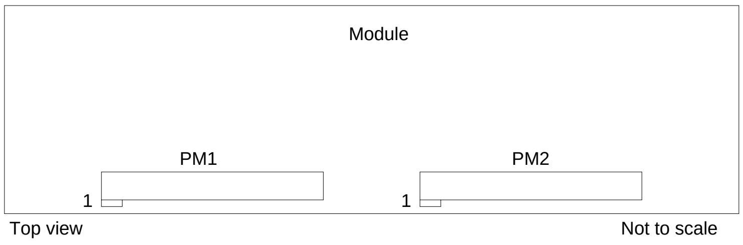


Figure 5.3.4. Module connector locations.

Board versions DS5101-04 and later do not have module connectors any more.

The DS5101-02 can host up to 8 modules containing two signal generation controllers each. Each module connector consists of two 50 pin sockets. Table 5.3.4 shows the pinout of the module connectors.

Left socket (PM1)				Right socket (PM2)			
Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal
1	/INT B	2	GND	1	D14	2	A0
3	/RES B	4	/CS B	3	D13	4	D8
5	VCC	6	/STRIG B	5	D12	6	D9
7	/TRIGIN A	8	GND	7	EXT0	8	EXT14
9	/TRIG2	10	/TRIGIN B	9	EXT11	10	EXT7
11	/TRIG4	12	/TRIG3	11	FLAG5	12	EXT13
13	/TRIG6	14	/TRIG5	13	FLAG3	14	EXT10
15	/TRIG8	16	/TRIG7	15	FLAG1	16	FLAG0
17	/TRIG10	18	/TRIG9	17	EXT8	18	EXT1
19	/TRIG12	20	/TRIG11	19	EXT9	20	EXT3
21	/TRIG14	22	/TRIG13	21	FLAG6	22	EXT2
23	VCC	24	/TRIG15	23	EXT15	24	EXT4
25	VCC	26	GND	25	FLAG2	26	EXT5
27	D3	28	D11	27	EXT6	28	FLAG4
29	D4	30	D2	29	GND	30	EXT12
31	D1	32	D10	31	CLK	32	GND
33	D0	34	D5	33	VCC	34	GND
35	D7	36	D6	35	EXTPIN A	36	PULLUP
37	A6	38	A9	37	n.c.	38	EXTPIN B
39	A7	40	A8	39	EXTIN A	40	EXTIN B
41	n.c.	42	n.c	41	n.c.	42	/CS A
43	A5	44	n.c.	43	/RES A	44	/INT A
45	A4	46	A11	45	SYNCEN	46	GA0
47	A10	48	A3	47	/RD	48	/WR
49	A1	50	A2	49	/STRIG A	50	GND

Table 5.3.4. Module connector pinout.

Name	Function
D0 - 14	15 bit data bus.
A0..11	12 bit address bus.
/TRIG2 - 15	Trigger outputs to other channels.
/TRIGIN A,B	Trigger inputs from other channels.
EXT0 - 15	Waveform outputs or external inputs.
FLAG0 - 6	Flags from flag register (FLR).
CLK	40 MHz system clock.
EXTPIN A,B	Signals from external connector.
EXTIN A,B	Buffered Signals from external connector.
PULLUP	Pullup signal from Setup register (STP).
/INT A,B	Interrupt outputs to Interrupt Register (IR).
/RES A,B	Reset inputs.
/CS A,B	Channel select inputs.
/STRIG A,B	Software trigger inputs from Software Trigger Register (STR).
/RD	Read signal.
/WR	Write signal.
SEL	Ram block select signal.
SYNCEN	Data update enable signal.

Table 5.3.5. Module connector signal description.

5.3.4 External clock connector (P37)

Pin	Signal
1	GND
2	Clock
3	GND
4	GND

Table 5.3.6. External clock connector.

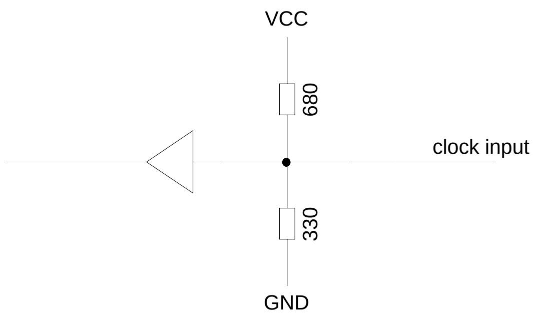


Figure 5.3.5 Interface circuit of the external clock input.

The external clock connector can be used to provide the DS5101 with an external clock in the range from 20MHz up to 40MHz. The input is terminated with a 680Ω resistor to VCC and a 330Ω resistor to GND. Make sure that the clock driver is able to drive up to 20mA. The clock source for the modules can be selected via the CLKSEL bit in the setup register (refer to section 5.1.4).

5.3.5 Board programming connector (P2, P3, P38)

The board programming connectors (P2, P3, P38) are for internal use only. Do not touch these connectors when power is applied to the DS5101.

5.4 DS5101 Data Sheet

Motherboard :

Number of channels	DS5101-04: 16 channels DS5101-02: 2 channels per module, max. 8 modules
Clock distribution	40MHz clock internal clock or external clock input in the range from 20MHz to 40MHz.
Digital I/O	TTL output buffer. DS5101-04: 10KΩ pull-up, pull-down or hold function (setup by jumpers) DS5101-02: 10KΩ pull-up resistor to VCC or GND (programmable common for all channels).
PHS-bus interrupts	interrupt from the modules and external reset.
PHS-bus interface	16 32-bit registers.
Physical size	340 mm x 125 mm x 20 mm Requires one full size length 8-bit AT-slot for power supply.
Power supply	+ 5V ± 5%, DS5101-02: max 6A via PC/AT DS5101-04: max 0.5A via PC/AT
Board temperature range	0 .. 70 ˚C

Module :

Number of channels	2
clock frequency	40MHz with on board clock or 20 to 40MHz with external clock
Resolution	25ns
Shortest output pulse	250ns
Memory	• Dual port memory for program (states) and data (delays). • Synchronization flag for update of complete sets of delay values 512 Delay values (30 bit) 128 State Command sets (90 bit)
Operations	• Output mode Set output Reset output Toggle output nop • Input mode Trigger on rising edge Trigger on falling edge Trigger on both edges nop • wait for a given time • Trigger other channels • generate PHS-bus interrupt • Loop counter • branch on software flag, state of other channel

Index